UNIVERSIDADE NOVA DE LISBOA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

A FRAMEWORK TO OPERATE TRANSFORMATIONS BETWEEN DISTINCT MODELS DIFFERING IN MODELLING CONCEPTS

POR:

JOÃO VASCO DA SILVA TRIGO VAZ

DISSERTAÇÃO APRESENTADA NA FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE NOVA DE LISBOA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

ORIENTADOR: PROFESSOR DOUTOR ADOLFO STEIGER GARÇÃO

PRESIDENTE: PROFESSOR DOUTOR ADOLFO STEIGER GARÇÃO – FCT/UNL

VOGAIS: PROFESSOR DOUTOR JOSÉ ANTÓNIO BARATA DE OLIVEIRA – FCT/UNL

PROFESSOR DOUTOR MANUEL MARTINS BARATA - ISEL

MESTRE PEDRO MIGUEL NEGRÃO MALÓ – FCT/UNL

LISBOA

DEZEMBRO DE 2010

AGRADECIMENTOS

Pretendo com este espaço dedicar um agradecimento a todas as pessoas que de alguma forma contribuíram para que esta tese acontecesse. Para todas elas, expresso desde já o meu sincero agradecimento. No entanto, pretendo dedicar um agradecimento especial:

Aos meus pais pela educação que me deram, por todo o apoio que sempre demonstraram e por terem feito tudo para que fosse possível concluir esta etapa da minha vida.

Ao Mestre Pedro Maló, pelo apoio, incentivo, experiência e mestria com que contribuiu ao longo da minha breve passagem pelo mundo da investigação, mesmo quando nem tudo correu da melhor forma.

Ao Mestre Bruno Almeida, não só pela sua amizade como por toda a sua objectividade, grande nível exigência e acima de tudo por ter dado o contributo certo no exacto momento para que este trabalho saísse do mundo do imaginário.

Aos meus mais próximos amigos com quem sempre partilhei, não só os bons mas também os mais difíceis momentos no decorrer do curso, e que sempre me conseguiram motivar sem nunca terem desrespeitado as exigências que o curso vinha exigindo.

Aos putos do GRIS, Buh!, Lois e Vivi, pelo excelente espírito de equipa e entreajuda que sempre criaram e que desencadearam em todos os momentos passados em conjunto.

Por fim, à Marta, pela sua amizade e amor, por ser sempre tão especial e inspiradora, por toda a paciência e dedicação demonstrada e por ter estado presente em todos os momentos decisivos ao longo do curso.

To me, finally...

ABSTRACT

With the demanding competitiveness of today's markets, enterprises must collaborate in an efficient manner in order to be profitable. However, today most large-scale information systems solutions in business are becoming strongly heterogeneous in their technical context, which ends up hindering interoperability. In this context, and to answer to some these interoperability issues emerged the Model Driven Interoperability (MDI). A key operation used in MDI is Model Driven Transformation, which aim at enabling systems to interoperate, leading to advantages such as flexibility and adaptability.

However, the execution of Transformations in between models evidence difficulties when handling scenarios of heterogeneity among systems. The main reason behind this is the lack of proper tools that can directly and efficiently provide us with the ability to execute transformations between models described in heterogeneous model concepts.

Therefore, and since currently there are already several tools offering excellent support to model-to-model (M2M) transformations, this thesis proposes to take advantage of all the features of those transformation tools in order to provide a framework that enables execution of transformations between model described in heterogeneous model concepts.

RESUMO

Com o aumento da competitividade nos mercados de hoje em dia, as empresas têm de colaborar de uma forma eficiente para que consigam ser lucrativas. Contudo, a maioria das soluções em sistemas de informação de larga escala estão a tornar-se cada vez mais heterogéneos no seu contexto técnico, o que acaba por levantar obstáculos à interoperabilidade. Nesse contexto, e para ultrapassar muitos desses problemas de interoperabilidade, surgiu o *Model Driven Interoperability* (MDI). Uma operação chave no MDI é a Transformação de Modelos, que tem como objectivo permitir que sistemas inter-operem, levando a vantagens como flexibilidade e adaptabilidade.

No entanto, a execução de transformações entre modelos levanta problemas quando se trata cenários que englobem grande heterogeneidade entre sistemas. A principal razão para que isso aconteça é o facto das actuais ferramentas de transformações de modelos estarem mais vocacionadas para transformações entre modelos descritos por meta-conceitos homogéneos.

Assim, e como já existem várias ferramentas que oferecem um excelente suporte a transformações de modelo para modelo, esta tese propõe que se aproveite as vantagens das características dessas ferramentas para que se desenvolva uma solução que permita a execução de transformações entre modelos descritos em meta-conceitos heterogéneos.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

With constant technology changes tends to get is difficult to maintain stable and reusable software assets, so, as a result, the network structure has emerged as a more flexible form for business arrangement, one that allow enterprises to quickly adapt to changes n the market environment. In (Miles, et al., 1995), a network structure is described as flat and highly flexible, controlled by market mechanisms rather than administrative procedures, where several organizations form an industry value chain according to their core competencies, obtaining complementary resources through strategic alliances and outsourcing.

Moreover, by coming together within networks, companies are able to cooperate dynamically and offer complex services, create new market opportunities, combine their knowledge, products and services, and jointly produce and offer new services and products (Nachira, et al., 2006). These enterprise networks highly efficient and dynamic are called Collaborative Networks (CNs), and form powerful mechanisms to improve competitiveness and agility in today's market environments (Camarinha-Matos, et al., 2006).

These CNs are normally composed of a group of legally distinct or related enterprises coming together to exploit a particular product or service opportunity, collaborating closely whilst still remaining independent, and potentially competing in other markets or even in the same market. Thus, collaboration is an intentional property that derives from the shared belief that together the members of a CN can achieve goals that would not be possible or would have a higher cost if attempted by an individual, and therefore it's considered a key strategic objective. (Camarinha-Matos, et al., 2005).

To cope with the demanding requirements of today's markets, the CN's members must collaborate in an efficient manner, which in its turn calls for more interoperation among its members resulting in a significant growth in the need for their systems to be interoperable, in order to behave as a single enterprise (Gunasekarana, et al., 2008).

That way, information integration is critical for efficient collaboration and to ensure coordination between members of the CNs, which is crucial to improve the performance and

competitiveness of the whole partnership. Information integration means that information is shared and exchanged efficiently, in a way that to each member is granted access to accurate and timely information that reflects the status of the overall partnership's business (Lee, et al., 2001) (Myhr, et al., 2005).

However, due to the diversity and most especially to the heterogeneity of systems, applications and technologies used among CNs by their members end up differing at many levels. Thus, achieving full integration between them is considered as an enormous barrier (Gray, 2003). For instance, observations state that 30% to 40% of an organisation only, $29 billion were spent by IT professional services (Gartner Group). This is mainly due to the fact that most of the systems used in CNs were not designed since the beginning to work in collaboration.

Most large-scale information systems solutions in business, science and administration today are becoming strongly heterogeneous in their technical context. With heterogeneity is meant, diversity and dissimilarity in type, structure, logical, and technical dependencies of their consisting parts (Kutsche, 1999). Having that in mind, and knowing that in a Collaborative Network is required efficient means of communication between systems, and that enterprises are becoming more and more different, there is the need to handle heterogeneity of many different, but logically related information resources.

The fact is that a CN is intrinsically a network of heterogeneous stakeholders (Park, et al., 2004). For instance, in a typical CN, multiple actors collaborate in doing a variety of activities (e.g. product development, product engineering, product manufacturing, product data management), and within each area there are multiple disciplines involved (such as Electronics, Hydraulics, Mechanical, IT hardware and software, and many others), which often refer to the same information but with different views, hence with different degrees of precision and detail. Further, the actors involved may be different (e.g. electrical engineers, mechanical engineers, CAD engineers, product data managers, etc.), and with different backgrounds (LI, et al., 2005).

Thus, this heterogeneity of people, processes, systems and disciplines hinders interoperability in a CN, originating not only technological but also conceptual barriers to interoperability (Chen, et al., 2007). The technological barriers stems from the multitude of software tools (e.g. CAD, CAM, DMU, PDM, CPC, etc) that is used by the different members

of the CN. Normally, in order to optimize the functions supported by them, these tools, use their own proprietary data models to structure the information according to their needs. Because each system has its own specific models, data is created and stored in multiple and incompatible formats, which ultimately results in syntactically and semantically inconsistencies when trying to exchange or share the data (Kim, et al., 2006).

These issues are currently being studied under the Interoperability domain. The IEEE defines interoperability as the capacity of two or more systems or components to exchange information and to be able to use the exchanged information. Therefore, it is not enough to exchange information, has to be created ways for that exchanged information to be used correctly, and that also applies to the exchange of services and functions.

As so, in the referred collaborative context where heterogeneous systems continuously grow the need for integration, the ability to interoperate is the key to overcome the barriers when exchanging information between heterogeneous systems. That way, Interoperability between systems is crucial for CNs to work efficiently between its members. That way, adequate mechanisms must exist to ensure that information can be created, shared and used where and when it is needed, throughout the most different situations.

These days, the most common solution to handle the described interoperability problems is to develop a direct translation between the systems that require exchange of information. However, these translators are extremely expensive to develop and maintain, since the required number of translators exponentially increases with the number of systems. Thus, is necessary to create a solution to solve these problems and that at the same time enables to reduce costs, and to increase reaction to environment and technology changes.

In this context, and to answer to some of those issues came the Model Driven Interoperability which is an emergent discipline that advocates the use of (software) models as primary artefacts of the software engineering process for interoperable systems. In addition to the initial goals of being useful to capture user requirements and architectural concerns, and to generate code from them, models are proving to be effective for many other engineering tasks. MDI is much more complex than simply defining a local serialization format, e.g., XMI. This would just resolve the syntactic (or “plumbing”) issues between models and modelling tools. However, interoperability should also involve further aspects, including behavioural specifications of models (which in turn describe the behavioural

aspects of the systems being modelled), and other “semantic” issues such as agreements on names, context-sensitive information, agreements on concepts (ontologies), integration conflict analysis (including for example automatic data model matching), semantic reasoning, etc.

MDI derives directly from Model Driven Engineering that was created to provide a solution to some specific problems, such as of the continual emergence of software technologies due to competition in the software industry that forces companies to port their software systems every time a new “hot” technology appears. As an example we can consider the evolution of middleware technologies. CORBA (OMG) is a widely accepted standard for middleware and many companies used it to implement their systems. We observe that Web Services partially replaces CORBA as middleware technology. Many existing systems may require porting to this new technology although their functionality may remain the same. When such a new technology appears it will cause the same problem of porting of existing systems. The constant changes in the technologies create a problem with portability which may require significant efforts.

MDE also aims at solving problems in current software development practices, caused by the fact that software development processes are driven by low-level design and coding. The maintenance and understanding of code is a difficult and error-prone process in case of large software systems. Other problems are introduced by the fact that large systems are not monolithic but modular. Different modules are built upon technologies suitable for the problem at hand. Therefore, software systems consist of components implemented in various technologies that need to interoperate. (Kleppe, et al., 2003)

1.1. CONTEXT

Interoperability is crucial to ensure efficient collaboration and synchronization among systems, therefore mechanisms must exist to ensure that information can be created, shared and used where and when is necessary. Thus, in order to cope with some of the interoperability problems described before, Model Driven Interoperability which derives from MDE, combines the principles that have been used in computer science and other engineering disciplines, of using modelling and models to develop software systems. Model

Driven Interoperability can be referred as being based and ruled by the principle that "*Everything is a model*" resulting in a clear separation between the phase of modelling and study of models, and the phase of realizing the models into the real world objects. Some models define the concepts that specify the constructs and the relationships that are used in a given modelling language, hence, making statements about what can be expressed in valid models of a certain modelling language. For now we'll call them "model concepts".

A key operation used in MDI is Model Driven Transformations. Model Driven Transformation is the process of operating one model into another model of the same system. A model may be transformed to multiple models that are functionally equivalent but still differing in the quality properties they possess. For example, one model may be more extensible while another model may be optimized for time performance. The aim of MDI is at automating model transformations as much as possible through the generation and evaluation of alternative transformations which are explicitly addressed in an MDI software development process. (Kurtev, 2005)

For instance, we shall take a step inside the MDI world and assume that two systems have their information described in models. Now model transformations between systems can be used to transform information –called data models in the MDI world– described conforming to one specific system into information conforming to another system.

Therefore, Model transformations provide critical foundation in granting effective and efficient communication among enterprises and leads to advantages such as flexibility, capability of changing with ease and to obtain a general view over the whole system. The consequence is that when one or more parts of a system need to be changed or replaced, other parts of the system won't have to be changed to avoid being affected in undesirable ways. In practice this means that in this context the cost of change gets more affordable. (Eriksson, et al., 2001)

1.2. PROBLEMATIC

Despite the great advantage of MDI, in the use of Model Transformations to handle some of the current problems of interoperability between systems, when dealing with heterogeneous systems two main problematic scenarios occur.

The first comes related to the fact that over the last years a large number of different models have been proposed which offer a similar core of modelling concepts, but tend to differ in some details since some models provide additional modelling concepts for enhancing expressiveness (Wimmer, 2008). In other words, data models expressed with different model concepts are difficult to compare and arises the problem of migrating data models expressed in one model concept to another. So it can be said that the first scenario arises when trying to transform data models described under two distinct and heterogeneous model concepts

The second scenario occurs when two data models, described by the same model concepts, have various heterogeneities between them, e.g., structural heterogeneities meaning that although semantically equivalent they are modelled differently. In general, to handle this (second) scenario, mappings (i.e., semantic correspondences between elements of two different data models) are the proposed mechanism.

Although being a recent approach, with already a large number of important model transformation tools (such as ATL, GREAT and more that will be discussed later with further detail), when applying model transformation that face the first scenario, there is a lack of proper tools that can directly and efficiently provide us with the ability to execute transformations between models described in heterogeneous model concepts. That is mainly due to technical limitations in both these tools and the transformation languages that these tools support, since currently model transformation tools are more targeted for executing transformations like the ones described in the second scenario rather than the ones described in the first.

Regardless of that and since currently there are already several tools offering excellent support to model-to-model (M2M) transformations, has to be developed a way to execute model transformations between heterogeneous model concepts like those referred in the

first scenario with the use of tools more dedicated at executing transformations between heterogeneous data models as in the second scenario

> In which way can homogeneous model transformations be applied in order to enable the execution of transformations in between models that are described using heterogeneous model concepts?

1.3. WORK APPROACH

The work approach for this work is based on the scientific method, composed by the following steps (Schafersman, 1994):

1. Definition of the problem;
2. Information Gathering and requirements;
3. Hypothesis formulation;
4. Experience perform;
5. Analysis of results and conclusions;
6. Publications.

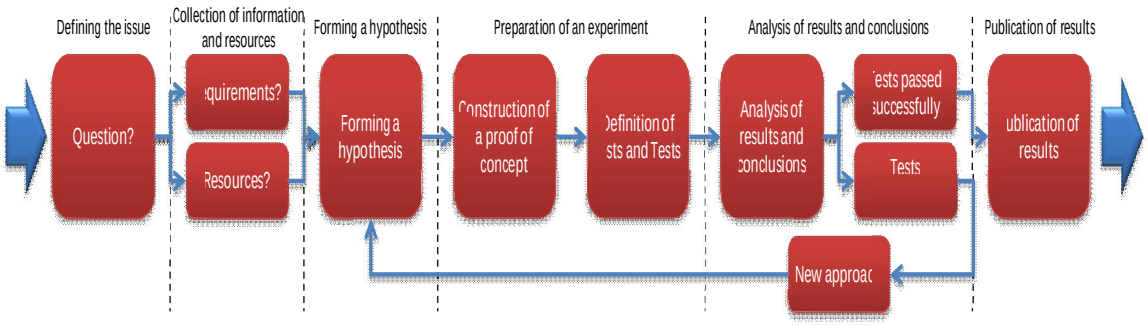

FIGURE 1 - RESEARCH METHODOLOGY

1. DEFINITION OF THE PROBLEM

The work starts by studying and analyzing the scenario, where some issues are identified leading to a question that drives the whole work. The identified problem is: In which way can homogeneous model transformations be applied in order to enable the execution of

transformations in between models that are described using heterogeneous model concepts?

2. INFORMATION GATHERING AND REQUIREMENTS

The requirements for the current work must be identified by a junction of the needs of the context and the research challenges. In the vision of industry, is necessary to evaluate the applicability of the solution in the real world.

So, in order to make this work is necessary to gather information about the requirements of this context and also gather scientific information about the existing technologies regarding model transformation engines.

3. HYPOTHESIS FORMULATION

Based on the information gathered and in previously defined requirements, is mandatory the formulation of a hypothesis. In this step is defined the conceptual realization that is the base for the analysis of the problem. So in this step a hypothesis to address the problem identified in the first step.

4. EXPERIENCE PERFORM

This step aims at evaluating the hypothesis through the use of the experimental implementation of the proposed solution as a proof-of-concept in order to gather information, like metrics, behaviours and other kinds of results. Then, the information gathered is evaluated facing the hypothesis. All the results must be taken in a controlled environment in order to minimize the sample's noise but also, control all the results of the experiment.

5. ANALYSIS OF RESULTS AND CONCLUSIONS

Facing the results obtained in the proof-of-concept against the hypothesis is possible to verify if the solution meets the defined requirements. The tests are applied in a controlled environment through experimentation to validate the solution. In case the tests fail, we start from step 3, where a new solution is proposed in order to successfully fulfil the defined requirements. In the end some conclusion are taken based on the obtained results.

6. PUBLICATIONS

The final phase of the scientific method is the publication and the exchange of the information gathered on the experiment. This exchange is critical not only to avoid the redundancy of the tests for the proposed hypothesis, but also for the evolution of knowledge and new formulations of new hypothesis based on the work done and reformulations and critics about the work.

A scientific article was published as a direct result of this work, with the title “Directives for applying model-driven transformations to heterogeneous models and modelling languages” which in the date of delivery of this document is in the reviewing phase of submission in Models 2010.

1.4. CHAPTER ORGANISATION

This thesis is divided in seven chapters:

1. **Introduction**: In this chapter, the business context of this work has been presented, describing the need for Model Driven Engineering and to one of its major activities, Model Transformations, and the major motivations for the realisation of this work are presented too. Lastly the research methodology makes the path to the research question as well as the goals and requirements.
2. **Background**: In the Chapter Two, the Model Driven Development (MDD) paradigm is described with a brief overview explaining its main foundations and principles, including a view on the importance of models. An alternative modelling approach based on Modelling Spaces is presented. To finish the chapter, the concept of Transformation Environments is detailed and its use is explained.
3. **Transformation Environments Study:** Chapter three introduces an analysis and comparison between current state of play Transformation Environments. Each one of them is described regarding some predefined requirements according to the established goals of the current work.
4. **Framework to operate transformations between distinct models differing in modelling concepts:** Chapter four proposes a conceptual solution to solve the problems described in the introduction. This is achieved with the help of the concepts and technologies presented in the chapters preceding this one.
5. **Solution Analysis and Validation**: This chapter describes the methodology used to analyse and validate the proposed solution. A Reference Implementation is suggested as well as some tests that help reaching a verdict.
6. **Conclusions and future work**: Finally in Chapter Six the conclusions and perspectives for future works are presented.

2. MODEL DRIVEN TRANSFORMATIONS

Model Driven Development (MDD), represents a promising software engineering approach to address software complexity, both by simplifying and formalizing (standardizing to allow automation) the various activities and tasks that comprise software life cycle (i.e. design, construction, deployment, operation, maintenance and modification). It is in fact an important paradigm shift in software development, consisting mainly in the application of models and model technologies, and its primary goal is to raise the level of abstraction at which developers operate, reducing both the amount of development effort and the complexity of the software artefacts that the developers use (Hailpern, et al., 2006) (Schmidt, 2006).

2.1. MODELLING APPROACH

Given today's increase of complexity in technology models are becoming more and more attractive as a powerful mechanism to precisely describe problems in a way that avoids delving into technological details, thus allowing a developer to focus on more abstract tasks and increasing productivity (Hausmann, et al., 2004). Moreover, another powerful feature of models is that they allow the problem to be described using terms and concepts familiar to people who work in the problem domain, rather than in terms only familiar to technology experts. This helps to bridge the gap between business and technology experts, (Cook, 2004).

Another key principle in MDD is the support of models at different levels of abstraction, from high-level business models focusing on goals, roles and responsibilities down to detailed use-case and scenario models for business execution. In order to support these principles, it's required very improved infrastructures, providing mechanisms to perform operations on models, ensuring consistency and traceability between them on the different levels of abstraction and from different viewpoints (Nordmoen, 2001).

In the context of MDD, a model is a set of statements about some System Under Study (SUS), i.e. it defines what elements can exist in the system. The statements provide an abstract representation of observed objects in SUS, i.e. abstract elements corresponding to real elements observed in the SUS and its domain of applicability (Seidewitz, 2003). A primary requirement to support the unification principle, is that, models must be formally

described using well defined (both, in terms of syntax and semantics) specification languages, which are suitable for automated computer interpretation. In this sense, the symbols and relationships that are used to construct a model of a given SUS in MDD, should be written in a well defined modelling language, which in its turn, is described by a metamodel. That is, a metamodel is itself a model that specifies the metaconcepts which are the constructs and the relationships that are used in a given modelling language, hence, a metamodel makes statements about what can be expressed in valid models of a certain modelling language.

Modelling languages help to specify, visualize, and document models, including their structure and design, in a way that meets all the requirements. Modelling languages such as Ecore or UML can be used for business modelling and modelling of other non-software systems too. Using any one of the large number of modelling languages available, one can analyze the future application's requirements and design a solution that meets them.

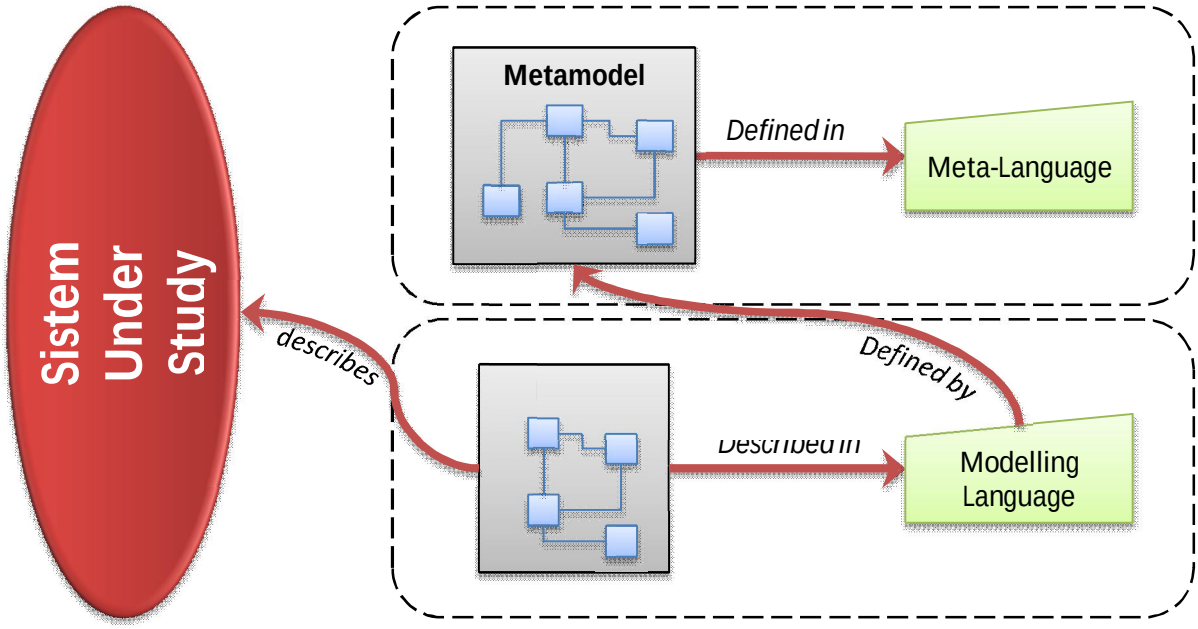

FIGURE 2 – SYSTEM UNDER STUDY

Nevertheless, a metamodel is also a model, hence it must also be written in a language (e.g. the Backus-Naur Form - BNF). This language is called a meta-language and it can be considered as a specialized language to describe modelling languages. In summary, one can say that, a model that describes a SUS is written in a modelling language defined according to the semantics provided by its metamodel, which in its turn is also written by a meta-language.

2.1.1. DOC N SHEMA

Models' concepts "conformTo" the metaconcepts that define them, in the sense that metaconcepts determine their nature, specify their precise meaning and form, and identify essential qualities. In a "n" layered modelling architecture, models at level "N" are said to "conformTo" the models in the level "N+1". In other words, the concept model at level "N" is defined by de metaconcept model in the level "N+1". In the same way the concept model at level "N-1" is defined by the metaconcept model in the level "N".

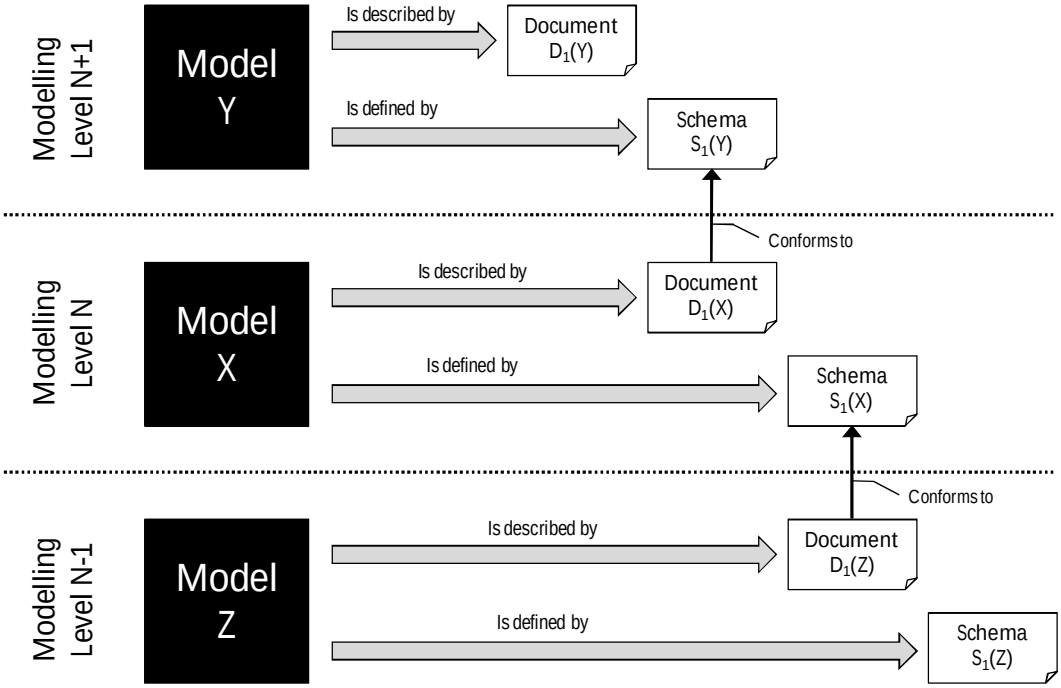

FIGURE 3 - DOCUMENT AND SCHEMA MODELING APPROACH

The modelling approach in Fig 3 shows that a model can simultaneously be described by a Document and defined by a schema. A schema is the way to define the structure, content and, to some extent, the semantics of documents. (Djuric, et al., 2006)

However, some models are only described by a document. In other words, there is no limitation saying that every model has to be at the same time a schema and a document. Nonetheless, it is mandatory that each document at level "N" must conform to the schema at level "N+1", unless we are referring to a reflexive model.

2.1.2. MLS N MS

A modelling space (MS) is a modelling architecture representing a specific model that can be described in various different modelling languages. Each language that describes the model

has a Meta Language Spaces (MLS). A model MS can have one or more Meta Language Spaces (MLS). MLS are represented as columns in Fig. 4, and are generally represented with four layers since they derive from the MDA-based architecture. MLS relies on the meta-concepts that describe the model. For instance, a specific model "Y" conforms to the meta-concept "Meta A0" which conforms to the meta-concept "Meta A1". For this example the MLS should be "A0.A1", however for the sake of the reader we assume that "A = A0.A1". The same is used for all the MLS.

MLS are based on a particular metamodel, call super-metamodel. A super-metamodel is a metamodel that is defined by its own concepts, and is placed in the topmost level of each MLS. If the super-metamodel was defined by some other concepts, it would not be a super-metamodel, and would exist at some lower layer in some other MLS. (model driven engineering and ontology development)

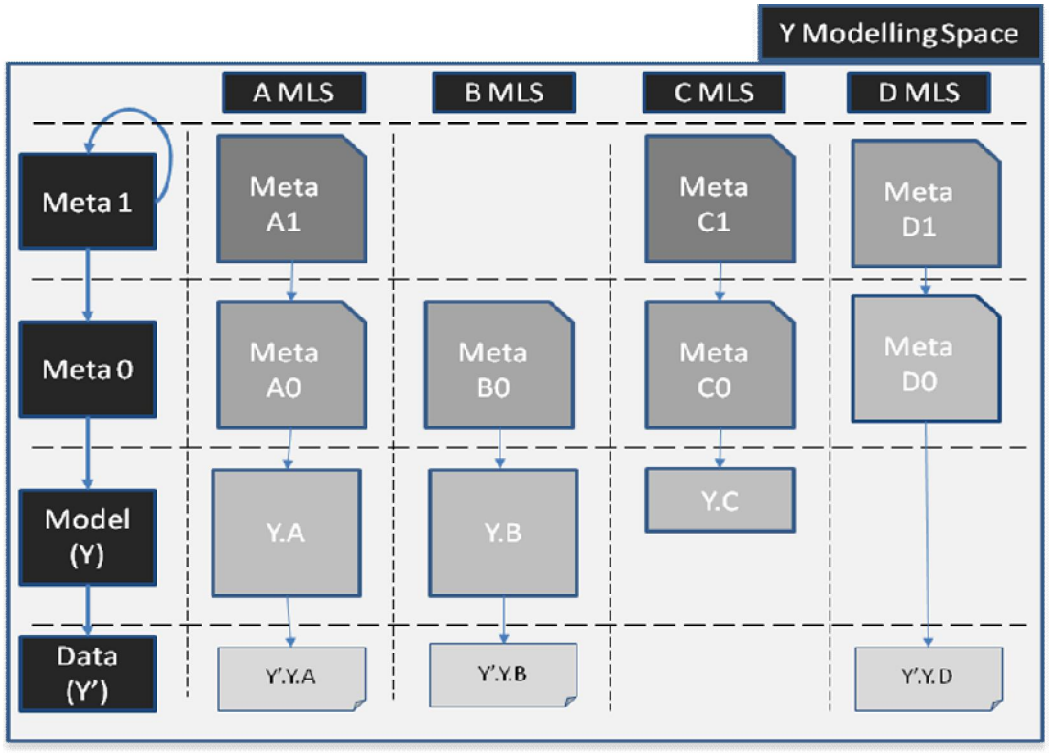

FIGURE 4 – DIAGRAM REPRESENTING MODELLING SPACE OF MODEL "Y"

The Fig. 4 represents the model "Y" Modelling Space (MS) containing four different MLS (A0.A1, B0, C0.C1 and D0.D1). In order to better understand the diagram, in the table below are defined what each shape means. The fig. 5 represents the shapes. The first type of shape (1) is used to express meta-concepts, which are usually called meta-models. In a MLS, the meta-concept placed in the topmost level is the reflexive meta-concept, the super-

metamodel described before. Each meta-concept describes models (2) from real world that will eventually describe data models (3).

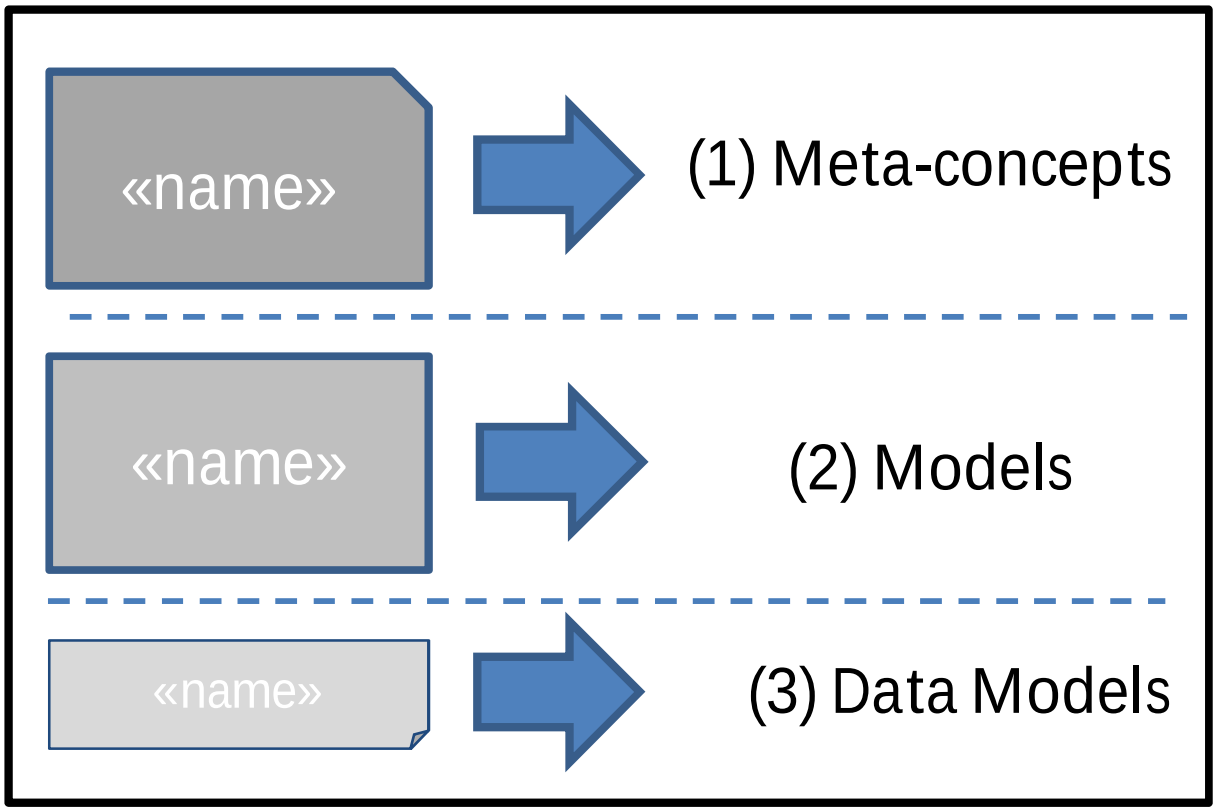

FIGURE 5 – SHAPE DEFINITIONS USED IN THE DOCUMENT

The first layer in each MLS represents data (3) that conforms to a model (2) in the layer above. Those models conform to the Meta-concepts (3) that are placed in the layers above. The MLS highest layer contains the reflexive super-metamodel. If the super-metamodel was defined by some other model concepts, it would not be a super-metamodel, and it would exist at some lower layer in some other modelling space.

To explain the diagram in Fig. 4 lets analyze the “MLS A0.A1” starting from the top. The “Meta” layers are pretty much straightforward as they are the meta-concepts defining the language in which de model is described, so “Meta An” means it is a meta-concept of the model.

The model “Y.A” indicates that there is a model “Y” described in the MLS of “A”. Below comes the data layer, “Y’.Y.A” indicating that data (Y’) conforms to the model “Y.A” above.

As discussed before, a model can be of two types. The first, where models are simultaneously defined by a schema and described by a document, are represented in Fig. 4

with the big rectangles that symbolize the aggregation of schema and document. In the second type models are only described by a document, which is represented with the smaller rectangles in Fig. 4 (like the model "Y.C"). It should be noticed that the position of the model "Y.C" placed on the top section of level "Model (Y)" is coherent with the position suggested for documents in the architecture shown in Fig. 3. It is also worth noticing that within each MLS, models that are only described by documents represent the lowest layer of their MLS.

"MLS B" suggest the existence of a reflexive model at layer "Meta 0", since "Meta B0" has no model in the layer above it means that it is defined by itself, so its schema is used to describe its document.

The example in "MLS C" represents a model language that is incapable of describing Data. In opposite, the "MLS D" lacks semantics notion, explaining the void at "Model (Y)" layer. Therefore, in which way can this architecture help to understand the path to achieve inter model operations between heterogeneous MLS?

2.1.3. MODELLING LANGUAGES

A modelling language results from an interpretation, i.e. a mapping, of the metamodel elements to its elements, such that the truth-value of the statements in the metamodel can be determined for any model expressed in the modelling language. Indeed, a valid model must not violate any statement deducible from its metamodel, which in this case, the model is said to be conformant to its metamodel. For example, the EXPRESS metamodel defines constructs like "Entity", "Type", "Where", etc, that are used in the EXPRESS language to create valid EXPRESS models (ISO10303-11).

In theory, unless a reflexive metamodel is used, the layers of models, languages, metamodels, and meta-languages can increase indefinitely. A metamodel is said to be expressed using the same modelling language that it intends to define, i.e. a minimal set of elements of the modelling language are used to express the statements of the metamodel. In other words, a reflexive metamodel is a self-describing model which self-conforms to its own semantics. Examples of reflexive metamodels are the Meta Object Facility (MOF), and Ecore, which has been introduced with the Eclipse Modelling Framework (EMF) (Budinsky, et al., 2004) (MOF08).

2.2. MODEL TRANSFORMATIONS

A particular pattern explored extensively in MDD is the use of model transformations. For instance, from a broader business-centred source model with possible abstract assumptions, a model transformation is normally used to produce a concrete target model in terms of the underlying implementation technologies. Normally, the transformation process uses a set of transformation rules, which are written in a special language, called the transformation specification, defining the changes that should be applied to a source model to produce a target model. This set of rules defines a mapping between the source and target models, which is then used by a transformation tool, to actually perform the transformation (Lin, et al., 2004). More specific details on how this process is realised are provided in this chapter.

A model transformation is an operation that modifies a source model into a target model. Nevertheless, the operations are intrinsically related, in the sense that, in order to perform a transformation, a mapping must always be defined, implicitly or explicitly. In fact, is the correspondence rules defined in a mapping that describes how a given model can be transformed into another model, thus the mapping is a pre-requisite for performing a model transformation.

A model transformation can be regarded as an operation that takes as input, a source model and a mapping specifying the model correspondences between the source and target model, and returns the target model as result. Depending on the type of mapping used as input of the transformation function, a model transformation can be classified as an instance transformation (i.e. a model instance mapping is used) or metamodel-driven transformation (i.e. a model type mapping is used).

Hence, a metamodel-driven transformation is a model operation that is performed by a transformation function, which receives the source model and the type mapping as input, and actually transforms the source model in to the target model, according to the correspondence rules between the source and the target metamodels (Richard, 1998). For instance, using the meta-modelling architecture, in a metamodel-driven transformation the mapping is established at the metamodel level, while the actual transformation happens at the model level. In an instance mappings is defined the correspondence between the elements of the model, which means that is defined how elements can be transform in one another.

The choice between the two categories of transformations, i.e. what type of mapping to use, depends on the desired level of flexibility when performing the model transformation operation. For instance, a metamodel-driven transformation is more deterministic that an instance transformation, since it uses a model type mapping which is based on the well defined types of the involved metamodels. Thus, if some additional information specific to the models involved (i.e. to the specific instantiations of the metamodels), is needed in the transformation process, then an instance transformation is more adequate, i.e. a model instance mapping should be used.

Additionally, another aspect that can influence the choice of which type of mapping to use, is whether the source and the target models are instances of the same metamodel or not, i.e. if the target and source Modelling Space share in fact a common MLS or not. In fact, a model transformation is further classified as endogenous, when the models (Modelling Spaces) are derived from the same metamodel (meaning MLS), or exogenous, when they are instances of different metamodels (meaning MLS) (Mens, et al., 2005). Model instance mappings are more suitable for endogenous transformations, since the source and the target models are expressed using the same type constraints defined in their common metamodel, thus there is no need to enforce further validation. Model type mappings, on the other hand, are more adequate for exogenous transformations since the rules and type constraints that define valid models are different for each model involved.

Nevertheless, sometimes a combination of the two types of mapping is often the best solution, since it allows the flexibility of the instance mapping, and at the same time, the type mapping can be used to preserve the type constraints defined in the metamodels that must be obeyed by valid instance models

2.2.1. Model Transformation Languages

Model transformations are specified using model transformation languages. A transformation language is a computer language designed to transform some input information described in a certain formal language into a modified output information that meets some specific goal.

There are already several proposals for model transformation specification, implementation, and execution, which are beginning to be used by Model-Driven Engineering practitioners. (Irazabal, et al.) The term "model transformation language" comprises all sorts of artificial

languages used in model transformation development including general-purpose programming languages, domain-specific languages (DSLs), modelling and meta-modelling languages and ontologies. Examples include languages such as the standard QVT, ATL and RubyTL.

These languages are specific to define model transformations; however an extra level of specialization can be realized on them. That is to say, we can define a transformation language specifically addressed to a given transformation domain. For example, we can create a language dedicated to the definition of transformations between data-base models or a language addressed to the definition of transformations between business models

Transformation languages can be of three types: Declaratives, Imperative, or Hybrid.

A transformation language is declarative if transformation definitions written in that language specify relationships between the elements in the source and target models, without dealing with execution order. Relationships may be specified in terms of functions or inference rules. Transformation engine applies an algorithm over the relationships to produce a result.

In contrast, an imperative transformation language specifies an explicit sequence of steps to be executed in order to produce the result.

An intermediate category may also be introduced known as hybrid transformation languages. These languages have a mix of declarative and imperative constructs. Usually, a transformation definition written in a hybrid language contains a set of rules that specify relationships between elements. Rules may have imperative bodies that specify the steps to be taken to produce the required result.

The reference standard in model transformation languages defined by the Object Management Group (OMG) is QVT. QVT operates in the layered MOF-based meta-modelling architecture prescribed by OMG. The abstract syntax of QVT is defined as a MOF 2.0 metamodel. This metamodel defines three sublanguages for transforming models. OCL 2.0 (Jouault, et al.) is used for querying models. Creation of views on models is not addressed in the proposal.

The three QVT languages collectively form a hybrid transformation language with declarative and imperative constructs. The languages are respectively named Relations, Core, and

Operational Mappings. These languages are organized in a layered architecture shown in Figure 6 .

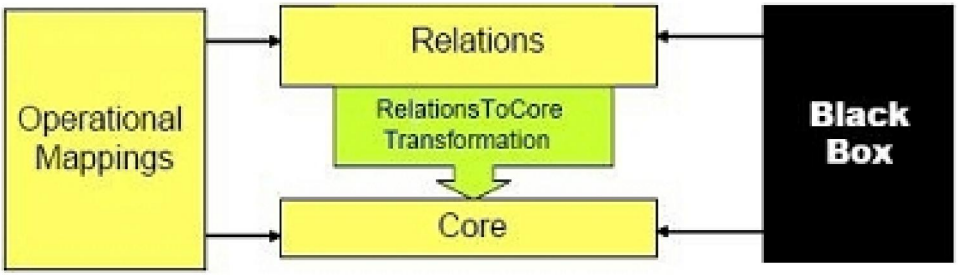

FIGURE 6 - QVT LANGUAGES ARCHITECTURE

The languages Relations and Core are declarative languages at two different levels of abstraction. The specification document defines their concrete textual syntax and abstract syntax. In addition, Relations language has a graphical syntax. Operational Mappings is an imperative language that extends Relations and Core languages.

2.2.2. TRANSFORMATION ENVIRONMENTS

Since we need not only a Transformation Language, but also a tool to execute the rules created in that language seems more appropriate to study the existing transformation Languages and engines as a technological pair, therefore we will call that pair a Transformation Environment. Thus a Transformation Environment is a technological pair, composed primarily by a Transformation Language and a Transformation Engine.

It is well known that to define model transformations is needed a model transformation language in which rules are created to map the transformation from one source model to a destination model. To execute these transformation rules we use an Execution Engine, which can interpret mapping definitions in order to perform model transformations.

However, a given language may be outstanding but have a poor engine to execute programs. The possibility to execute programs on a more mature and optimized engine originally designed for another language could always improve performance.

Some languages may not even have an engine. For instance, at the time of writing of this work the QVT Core language is not yet supported by an engine. Moreover, a given model engineering platform may only provide a QVT engine to comply with the standard. Some users may still want to use another language on top of this platform.

2.3. SUMMARY

In this chapter is provided some background focused on Model Driven Transformations that will support the rest of the work in the thesis.

Firstly is described a Modelling Approach that is based in two theories. First that each model can be described in a document and defined in a schema, and second, that each model has a Modelling Space (MS) that includes a Meta Language Space (MLS) for each language in which the model is described.

Lastly, some notions are given around the model operations paradigm, in special model transformations that will require extra attention through the rest of the document. To finish the chapter, Transformation Environment definition is presented has an aggregate of technologies that work as a whole, thus not making sense to compare them as an independent technologies.

3. TRANSFORMATION ENVIRONMENTS STUDY

Despite the fact that no model transformation language is clearly better than another, still some languages are more suitable for specific tasks compared to others. In general, if the transformation to implement includes structurally similar metamodels, declarative approaches tend to be usually good choices. This is due to the fact that a declarative approach is more abstract and therefore supports fewer reserved keywords, resulting in a lower number of code statements, but only when the transformation is not complex (Puder, et al.). Metamodels which differ structurally, where concepts from one metamodel do not have direct equivalents in the other metamodel, are more complex to transform. For such situations imperative approaches are more suitable because they are more expressive than declarative ones. Hybrid approaches on the other hand claim to combine the advantages of both imperative and declarative approaches. Hybrid approaches are therefore most suitable for transformations which include a certain amount of structural similarity, but still include concepts which cannot be mapped to each other directly.

In the end, the most appropriate model transformation language has to be chosen depending on the requirements of the given transformation goal and the infrastructure of the MDE environment used. In order to support our decision and since there are several interesting Transformation Environments available, we will study and classify them according to a series of requirements that will help to make the most adequate choice possible. Consequently, we start this chapter by defining the set of requirements which the Transformation Environment needs to accomplish in the view of the proposed objectives, followed by a description of some Transformation Environments, and to finish, a comparative analysis of the proposed technologies.

We will focus mainly on more practical aspects of each TE (such as Supporting Community, License, Complexity, etc.), that way we will not analyze deeply the technical properties of each TE, (such as syntactic separation, rule application strategy, rule scheduling, application condition, reflection, etc.) since we assume that all of the TE described here could achieve our goals.

3.1. REQUIREMENTS AND CRITERIA

In order to define which Transformation Environment (TE) best fits our purposes we have to define the requirements that such technologies must have. Each requirement has its own way of classifying the Transformation Environment (TE), that way and so that completely different requirements can be put together each requirement was given a weight (P) corresponding to its importance to the current work. As an example, it's far more important for the Transformation Environment (TE) to be free and open source, than to support bidirectionality since for directionality there is a turn around, which is to develop a new set of rules for the opposite direction, whereas for closed licenses there isn't. Thus, all requirements will be rated with a scale of three points, where 3 (three) means High important, 2 (two) Medium importance and finally 1 (one) yields Low importance.

After classifying each TE regarding the requirements described next, a result is computed by using the following formula:

$$= \frac{\sum_{\dots} (\quad)}{\sum_{\dots} (\ )}$$

Where:

- "P" represents the weight of each requirement and
- "Z" the amount of points that the TE received in that particular requirement

The weight among the requirements defined below is extremely specific to this work, and it is highly subjective since it derives directly from the author's opinion, but is the one that most closely addresses the needs of the proposed work. Therefore, the six chosen requirements are *License, Supporting Community, Simplicity, Execution Time, Development* and *Bidirectionallity*.

Since the solution being presented with this work is intended to be made available for other developers to continue the work done so far and to adapt current functionalities to their needs was given the maximum weight (3) to the License Requirement. Given that we're dealing with a recent approach to software development - Model Driven Development -, having a good supporting community is crucial to determine the success of a technology.

That way, a weight of three (3) was given to the Supporting Community requirement. Although creating transformation rules in some technologies can become a very difficult task, the Simplicity requirement ends up being extremely subjective since some people might find it easier to work with some technologies, whilst others could prefer different technologies. Therefore, we will base this point on our preferences and a weight of two (2) was set. The execution engine requirement isn't as crucial as the other points discussed, therefore we chose to give a weight of one (1). Although keeping up with evolution is important, so that execution engines can, for instance, work with new modelling languages, it would be much more crucial if we were dealing with the development of a commercial tool. Therefore, we chose to give a weight of two (2) to the Development requirement. Although important, Bidirectionallity acts more as a work saver, since developers will only have to develop one transformation rule. Thus, it is given a weight of one (1) to this requirement.

To better understand the meaning of each requirement a brief description follows:

License

Free refers to the freedom to copy and re-use the software, rather than to the price of the software. Open-source software (OSS) is computer software that is available in source code form for which the source code and certain other rights normally reserved for copyright holders are provided under a software license that permits users to study, change, and improve the software. Since this work is being done within the field of research for academic purposes is essential the use of free and OSS.

However, if for instance the work is applied to the industrial "world", it is, too, of extreme importance that the technology is free and open source to keep costs low and at the same time to enable enterprises to connect and customize their systems, social tools and web services. For instance, within a Collaborative Network tend to coexist a wide range of enterprises, from small to big corporations. Therefore, if we aim at solving some of the actual interoperability problems among CNs, we have to make our solution is affordable even for the low budgeted enterprises.

To classify each TE with this requirement we gave one point if the TE is free and another point for being open source.

Supporting Community

Having a large, well organized supporting community provides a valuable feedback that will help tracking bugs and boosting improvements or even new features, especially if dealing with open source software where users can commit their work which will improve the software as a bundle.

It is especially useful for complex technologies since it helps decreasing the learning curve time for new users whose questions are answered by experienced users in forums or newsgroups.

To classify each TE we used three points that we'll be distributed depending on how active the supporting community is. For instance, if the community surrounding the tool/project is inexistent we give only one point, whereas if the community gives some exceptional support we give three points.

Simplicity

The more complex technologies are, the more difficult it turns out to develop a new solution, especially for new users. Therefore sometimes technologies which are easy to understand but less powerful might end up helping users to develop better solutions than complex but more powerful technologies. That is usually due to the fact that users can't use the full potential of complex technologies unless they are already extremely experienced at using it.

To classify each TE, and since this requirement is subjective, we used three points that will be given according to the author's opinion based on his experience. Three points are given for the easiest and simplest technology, and one point to the most difficult to be used.

Execution Time

Some technologies can only be executed in design time, limiting the whole solution especially if it is pretended to execute the solution in runtime embedded in other applications. That is important when the technology by itself cannot handle all the established objectives for the solution, meaning that it needs to run inside other application designed for other purposes.

One major advantage of executable (in runtime) software process models is that once defined, they can be simulated, checked and validated in short incremental and iterative cycles. This also makes them a powerful asset for important process improvement decisions such as resource allocation, deadlock identification and process management. (Bendraou10b)

To classify each TE we used a Boolean answer (yes/no) to the question of whether it can or not be executed at runtime. We give one point for a “yes”, and no points for a “no”.

Development

Technologies that are continuously under development by the creators tend to keep up with the evolution of their specific area of knowledge. This is done through the launch of new versions to support new features, or updates to solve some bugs. Working with an obsolete technology is never a good option unless there are no viable alternatives. Generally when a technology is obsolete means it stopped being developed at some point in time.

To classify each TE for this requirement we used three levels depending on the stage of development: Stopped, Under development, and Mature. Corresponding respectively to zero, one and two points.

Bidirectionallity

Bidirectional model transformation is an important requirement which describes not only a forward transformation from a source model to a target model, but also a backward transformation that reflects the changes on the target model to the source model in a way that consistency between two models is maintained. (Hidaka, et al., 2008)

Given the need for transformations to be applied in both directions, there are two possible approaches: write two separate transformations in any convenient language, one in each direction and ensure that "by hand" that they are consistent, or use a language in which one expression can be read as a transformation in either direction. (Stevens, 2007)

We will classify each TE with a Boolean (yes/no) option, giving one point for "yes" and zero for "no".

3.2. TRANSFORMATION ENVIRONMENTS

In this section we describe briefly the Transformation Environments technically and according to the defined requirements. The list of Transformation Environments is not exhaustive, since our goal is to cover transformation languages with available execution engines with a satisfactory level of maturity.

3.2.1. ATL

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
ATL	●/●	●●●	●●●	Yes	Mature	No

ATL is inspired from the OMG QVT standard recommendation and builds upon OCL formalism. ATL is a hybrid transformation language, meaning it contains a mixture of declarative and imperative constructs. ATL transformations are unidirectional, operating on read-only Ecore models and producing write-only target models. ATL execution engine is based on a virtual machine architecture.

License: ATL is based on the EPL (EPL) open source agreement

Supporting Community: ATL has a great supporting community that provides a valuable feedback that helps tracking bugs and boosting improvements or even new features; Moreover, it helps reducing the learning curve time for new users whose questions are answered by experienced users in forums and newsgroups.

Simplicity: ATL is build upon the heavy use of OCL, which is extremely well-supported and expressive; Uses a simple declarative language, thus the programs are concise that makes it easy even for non-programmers to create solutions; Code is often in the helpers and not in the actual transformation rules;

Execution time: Compiled transformations generate an "asm" file that can be executed in run-time with the use of ATL VM. It can also be executed in design time with ATL IDE.

Development: ATL is a mature technology, which is still being supported by developers. The last version dates from June 2010.

Bidirectionallity: ATL only supports one way transformations, therefore is unidirectional;

3.2.2. TEFKAT

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
Tefkat	•/•	•	••	No	Stopped	No

Tefkat (TEF) implements a state-of-the-art declarative, logic-based model transformation language DSTC which is suitable for Model-Driven Development (MDD) and data transformation. It is implemented as an Eclipse plugin that leverages the Eclipse Modelling Framework (EMF) to handle models based on MOF, UML2, and XML Schema. Tefkat has a simple and familiar SQL-like syntax, is specifically designed for writing scalable and re-usable transformation specifications using high-level domain concepts rather than operating directly on XML syntax. There are several aspects in the language worth noting:

The Tefkat engine is a Java-based implementation of the XMorph (Xmo) transformation language, which uses the Eclipse Modelling Framework (EMF) as its modelling environment. The engine is suitable for standalone use and is inviolable from the command-line. The engine as a whole comprises approximately 4000 to 5000 lines of Java code.

License: Tefkat is based on Eclipse, which is a powerful free and open-source platform;

Supporting Community: Tefkat has weak documentation, presenting only some tutorials and lacks in a strong community.

Simplicity: Tefkat has a simple and familiar SQL-like syntax, is specifically designed for writing scalable and re-usable transformation specifications using high-level domain concepts; Transformations do not specify a traversal order of the input models, nor an execution order for the rules – implementations must ensure that rules are executed in an order that satisfies the semantics;

Execution time: Can only be run in design time;

Development: Tefkat is a mature technology, but has stopped being supported by developers. The last version dates from January 2008.

Bidirectionallity: Tefkat only supports one way transformations, therefore is unidirectional;

3.2.3. MEDINI QVT

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
Medini QVT	-/●	●●	●●	Yes	Mature	Yes

Medini QVT (ikv) is a tool set for model to model transformations. The core engine implements OMG'S QVT RELATIONS STANDARD, and is licensed under EPL. The Relations language provides capabilities for specifying transformations as a set of relations among models. Relations contain a set of object patterns. These patterns can be matched against

existing model elements, instantiated to model elements in new models, and may be used to apply changes to existing models.

MediniQVT also includes tools for convenient development of transformations, such as a graphical debugger and an editor.

License: Free for non commercial use and open source;

Supporting Community**:** Some documentation is available but lacks a good supporting community.

Simplicity: Execution of QVT transformations are expressed in the textual concrete syntax of the Relations language;

Execution time: Can be run inside a Java application to execute a transformation or in design time with the medini IDE;

Development: Medini QVT is a mature technology that is still being supported by developers. The last version dates from March 2010.

Bidirectionallity**:** Bidirectional transformations are supported;

3.2.4. SMARTQVT

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
SmartQVT	•/•	•	•	Yes	Stopped	No

SmartQVT (Sma) is developed by France Telecom R&D. It is written in Java and available under EPL open source license. SmartQVT aims to implement the QVT-Operational language, which is a subset of QVT standard. Thus, it follows the imperative model transformation approach, which is dedicated to express model-to-model transformations.

SmartQVT acts as a compiler in that sense that given QVT-code is compiled to Java source code. To accomplish this, the QVT parser converts QVT textual syntax into corresponding representation in terms of the QVT metamodel. Then, the QVT compiler produces, from a QVT model, a Java program on top of EMF generated APIs for executing the transformation. Because of this architecture, it is possible that the SmartQVT compiler can be used in connection with other tools which output a QVT model conforming to the QVT metamodel. Additionally, serialized QVT transformations conforming to the QVT metamodel can be loaded and executed in runtime.

License: available under EPL open source license, thus is free software;

Supporting Community: Some documentation is available but misses a good supporting community;

Simplicity: Is hard to learn, but also more expressive because of its complex imperative approach;

Execution time: Serialized QVT transformations conforming to the QVT metamodel can be loaded and executed in runtime;

Development: SmartQVT is not being developed anymore;

Bidirectionallity: Because of its imperative approach, SmartQVT basically only supports unidirectional model transformations.

3.2.5. GREAT

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
GReAT	-/●	●	●●●	No	Mature	No

Graph Rewriting and Transformation (GRE) is a language that allows users to specify graph transformation in a graphical form with formal and executable semantics. GReAT is based on

the theoretical work of graph grammars and transformations and belongs to the set of practical graph transformations systems. GReAT have a declarative rule language and an imperative language for the rule application order. The transformation rules are specified by drawing patterns of UML classes from elements of the meta-models, and the rules are sequenced to form a transformation program. As a whole, we assume these languages to be hybrid.

GReAT can be divided into three parts:

- Domain specification and heterogeneous transformations
- Graph transformation language
- Control flow language

The transformation is executed using the GR-Engine that provides the fastest way of testing transformation prototypes. The transformation rules are interpreted by the engine and thus the level of performance is not as high as with compiled code. The engine is often used early in the development transformation so that results are tested quickly. (89-252-1-PB)

License: Free for non commercial use and open source;

Supporting Community**:** Some documentation is available but lacks a good supporting community.

Simplicity: Easy and fast for specifying simple model transformations, however some features require precise definition of the semantics;

Execution time: Can be run in design time;

Development: GReAT is a mature technology, that is still being supported by developers. The latest update dates from April 2010.

Bidirectionallity**:** GReAT only supports one way transformations, therefore is unidirectional

3.3. COMPARATIVE ANALYSIS AND CONCLUSION

Now that we described in detail the group of transformation environments that most suits us we will apply the formula to help us reach a conclusion.

The table below summarizes the characteristics of each Transformation Environment described regarding the requirements previously defined.

TABLE 1 - SUMMARY OF THE POINTS GIVEN TO EACH TE

Req.\Tech.	License (P=3)	Supporting Community (P=3)	Simplicity (P=2)	Execution time (P=1)	Development (P=2)	Bi-directionallity (P=1)
ATL	●/●	●●●	●●●	Yes	Mature	No
Tefkat	●/●	●	●●	No	Stopped	No
Medini QVT	-/●	●●	●●	Yes	Mature	Yes
SmartQVT	●/●	●	●	Yes	Stopped	No
GReAT	-/●	●	●●●	No	Mature	No

With the help of the table we can now easily execute the formula referred in section 3.1 to each TE, yielding the following chart that represents the total points gathered by each TE:

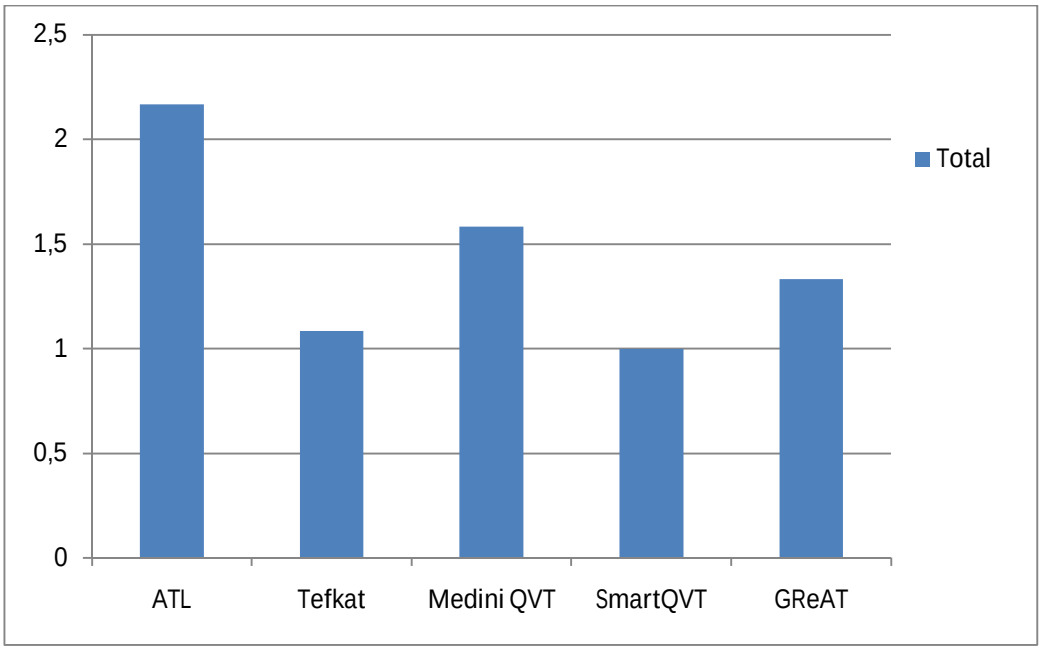

FIGURE 7 – CHART WITH THE FINAL RESULTS

Analyzing the chart in fig. 7 comes clear that ATL is the Transformation Environment that is most adequate to our predefined requirements. This is due to its hybrid approach, which by incorporating declarative and imperative constructs, makes ATL the most balanced in this evaluation. Most features which are needed for an average model transformation are implemented. Besides that, ATL is relatively easy to learn, because it just offers the most needed model transformation features in a relatively straightforward way, and on other hand leaves out more sophisticated features. ATL is able to cover a broad territory of model transformations, but complex transformations are harder to implement.

Moreover, ATL has a great supporting community which provides a valuable feedback that helps tracking bugs and boosts improvements or even new features. This helps reducing the learning curve time for new users whose questions are answered by experienced users in forums and newsgroups.

ATL is based on the Eclipse Platform, which is a powerful open-source platform whose evolution and adoption rate have been growing at a very fast pace, and whose flexibility, extensibility, ease of use, etc. make it an ubiquitous platform. Eclipse provides a lot of modelling-related functionality which includes EMF, as a sound meta-modelling infrastructure, or the MDT-OCL project, as an implementation of the OCL language.

MediniQVT, which comes classified in second, has the potential to solve many transformation problems in an elegant and efficient way. In fact, speaking of QVT in general, the problem adverts from the fact that it takes some time to understand and be able to use all QVT features, which allied with a weak supporting community sentences prematurely the use of the discussed QVT Transformation Environments.

3.4. SUMMARY

In this chapter we focus on studying Transformation Environments. In order to study and compare them, some requirements are defined in the most convenient way for the current work.

Each of the TE under study is described carefully focusing mainly on more practical aspects, such as Supporting Community, License, Complexity, and more.

After each Transformation Environment is classified, a comparative analysis of all the Transformation Environments is done in order to reach a conclusion regarding the one that most fits our needs.

4. FRAMEWORK TO OPERATE TRANSFORMATIONS BETWEEN DISTINCT MODELS DIFFERING IN MODELLING CONCEPTS

Modelling Spaces (MS) are often described in various Meta Language Spaces (MLS). Despite that, different MS end up being described in heterogeneous MLS, meaning that some MS don't share a common MLS between them, which difficult the exchange of information between MS. In this chapter are provided guidelines to overcome this problem.

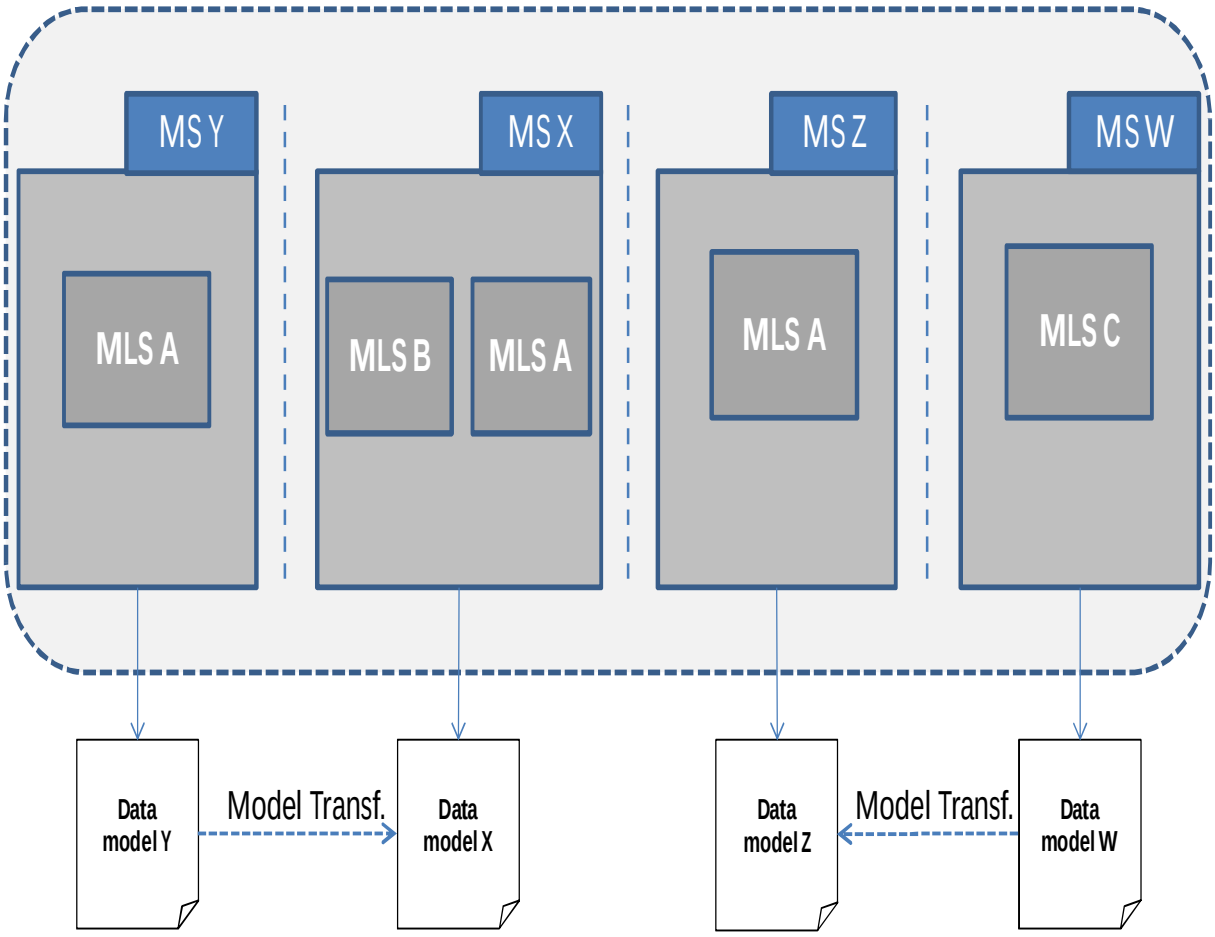

FIGURE 8 – EXAMPLE OF POSSIBLE MODEL TRANSFORMATION

Analyzing the Figure 8 above, two issues arise: MS Y with "model Y" wants to transform data into MS X throughout "model X"; System W wants to transform data with System Z throughout "model Z";

Therefore this section gives directions on how to perform model-driven transformations between heterogeneous models in two different situations: when both Modelling Spaces have heterogeneous MLS and when both Modelling Spaces share a common MLS.

To better understand the proposed solution, its concept was divided into two main points:

- Achieving model transformations between different Modelling Spaces that are described using the same MLS. Heterogeneous Models, with non-heterogeneous MLS.
- Achieving model transformations between Modelling Spaces that have different MLS describing their models. Heterogeneous models and heterogeneous MLS having: "Nb=Na" and "Nb≠Na" where "Na" represents number of layers in "MLS A" and "Nb" represents number of layers in "MLS B".

4.1. TRANSFORMATIONS BETWEEN HETEROGENEOUS MS WITH THE SAME MLS

The first point provides guidelines for the most essential step in the whole work, since it represents the way in which two different MS, which represent different models, can inter-operate if described under the same meta-language, in other words, described in the same MLS.

Therefore, at this stage the two heterogeneous Modelling Spaces (which are called that way since they both represent two heterogeneous models) share a common MLS. This means they share the same meta-concept levels, which is, as seen in the previous chapters, the only possible way to execute Model Transformations using the common Transformation Engines.

The two Modelling Spaces “MS X” and “MS Y” are specified in the Fig. 9 below.

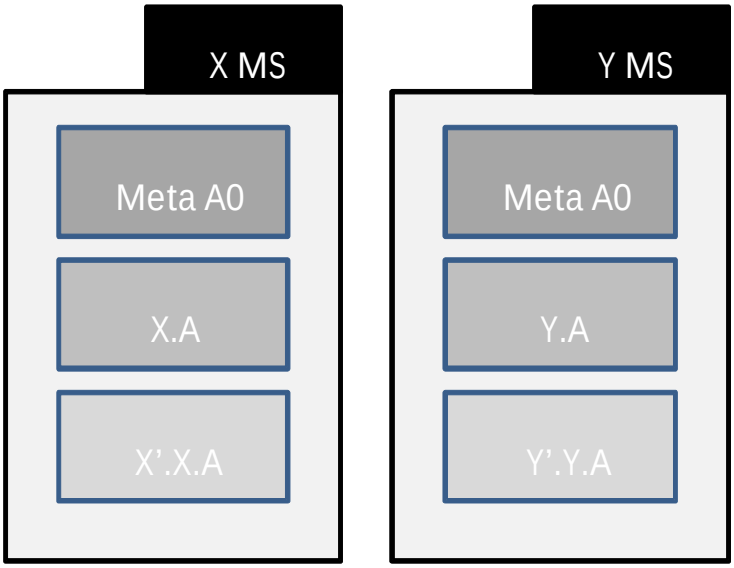

FIGURE 9 - X AND Y MODELLING SPACES

As seen before and since both Modelling Spaces share the “MLS A”, means that in practice both models are described under the same meta-concept that in the referred example is the “Meta A0”. That meta-concept is the so called common base between the two Modelling Spaces.

That way, mappings can occur between models “X.A” and “Y.A”, since they are described under a common meta-concept. As a result, “X’.X.A” data is transformed outputting “Y’.Y.A” data. This means that according to the correspondence rules between the source and target model-concepts, the source data model is transformed in to the target data model conforming to the model-concept “Y.A”.

In Fig. 10 the model Transformation process is detailed:

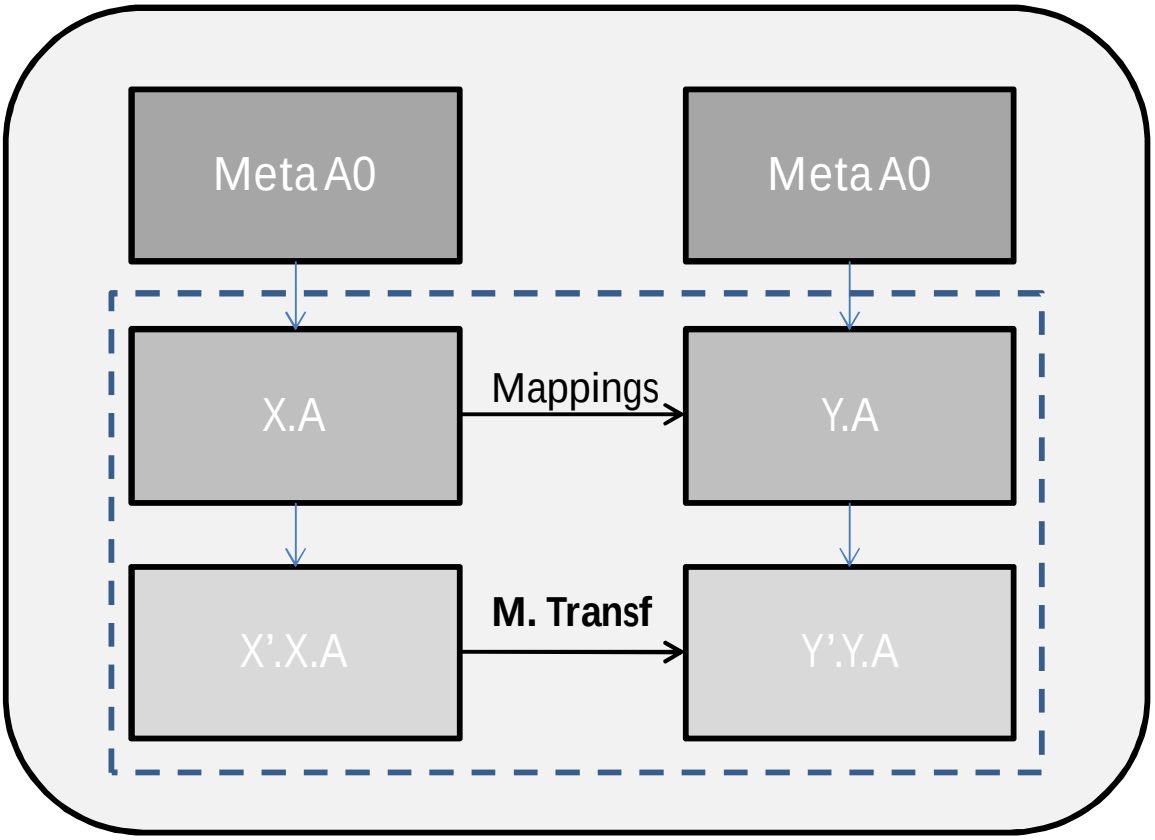

FIGURE 10 – MODEL TRANSFORMATION DETAILED

Although it may look pretty straightforward, due to the limitations on the existing technologies, executing model transformations between models with a common meta-concept is a good way to enable heterogeneous Modelling Spaces to exchange information.

4.2. TRANSFORMATIONS BETWEEN MS WITH HETEROGENEOUS MLS

In the second point both MS have heterogeneous Modelling Spaces, that don't share a common MLS. Which means that must be executed a model transformation between models (in different MS) described in heterogeneous MLS.

Heterogeneous MLS means they differ in their "Meta" levels in a way that to accomplish model-driven transformations between them is required some additional steps due to the limitations of the existing transformation engines. The process was divided in:

- Defining a common MLS between MS
- Transforming each Modelling Space MLS to the chosen common MLS.
- Executing the final model transformation

To better understand the idea described below, we will support the solution concept description with an example.

A MS that works with model "X", which is detailed in Fig.11, is originally described in the "MLS B". The same model "X" can also be described under the "MLS A", just like Fig. 11 shows.

Now, that same MS will want to transform data information to a MS describing the model "Y" (like the one shown in Fig.10) in its "MLS A". We can easily conclude that both systems are using different MLS to describe their models.

So, how can both MS execute a model transformation if they are working in different MLS (one in "MLS B" and the other in "MLS A"? The answer to that question leads us to the first step of the solution concept.

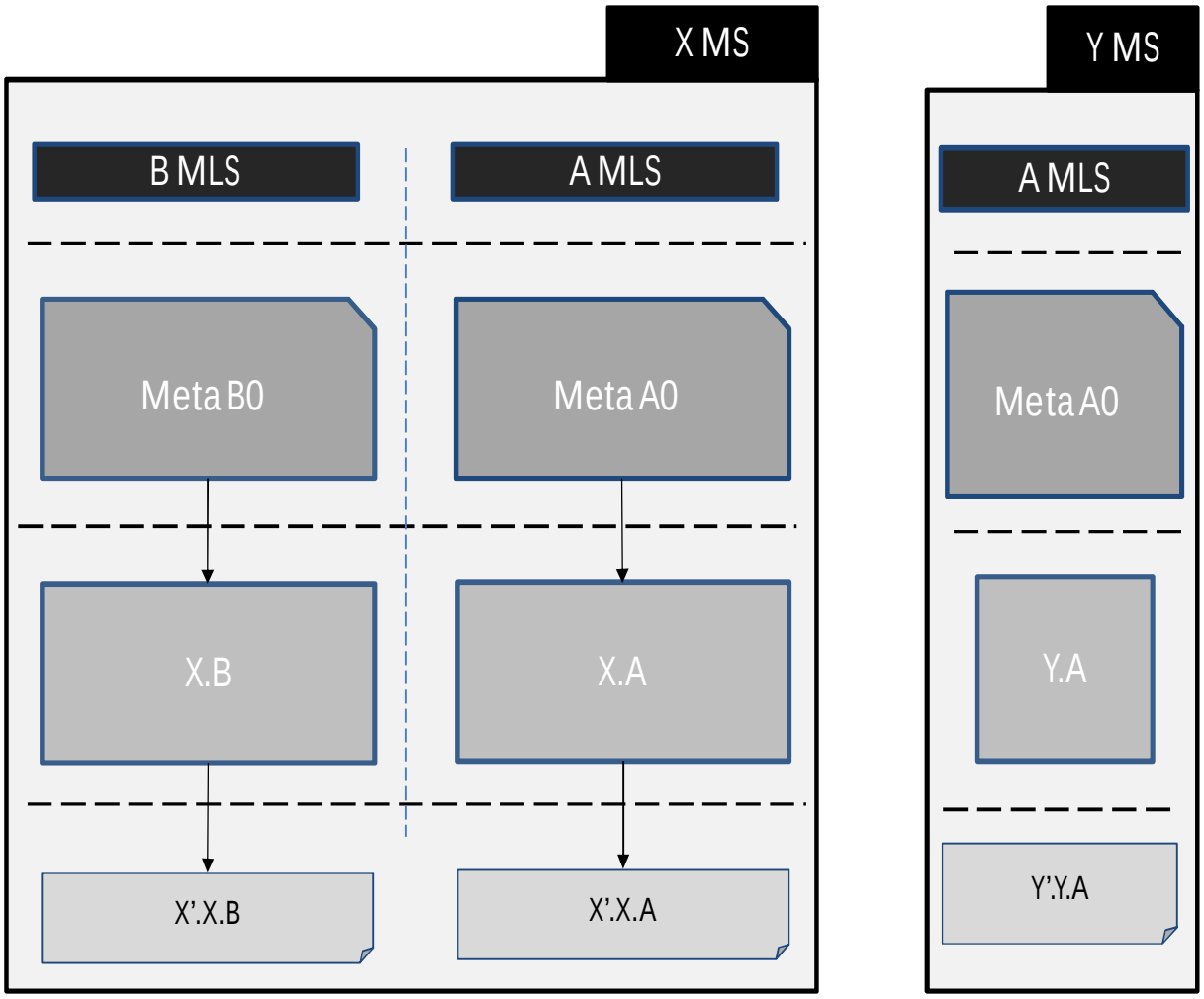

FIGURE 11 - "X" AND Y MODELLING SPACES

Step 1- Defining a common MLS between both MS

Therefore the first step is to analyse each model Modelling Space to conclude in which MLS a model can be described. The whole purpose behind that analysis is to achieve a common MLS between both MS so that mappings between the two MS can be accomplished.

Thus, analyzing "MS X" and since model "X" can both be described in "MLS A" and in "MLS B" and "MS Y" is only described in "MLS A", the "MLS A" is defined as common base for the transformation of models from "MS X" to "MS Y".

So, the MS that wants to send information has to transform its data described in "MLS B" to data described according to "MLS A" in order to achieve a common base with the MS working with "MS Y".

Step 2- Transforming the chosen common MLS (achieving the common MLS?)

This brings us to the second step. How to transform model "X" described in "MLS B" into a model "X" described in "MLS A"? With the use of Model Transformations (MT). However "MLS A" and "MLS B" are heterogeneous (they differ in their super-metamodel) and model-driven transformations using the common transformation engines are only possible when models share the same super-metamodel. Therefore it's not possible to directly transform models described in "MLS B" to models in "MLS A" with a model-driven approach.

So has to be found a link to enable transformations from "MLS B" to "MLS A". At the same time we must have the capacity of describing the model "X" in its original form. Therefore, to grapple the problem we introduce an auxiliary MLS that will enable to emulate the needed transformation from "MLS B" to "MLS A" with the help of some additional steps.

The auxiliary MLS is named "MLS B.A" since it both describes "Meta B0" and conforms to "Meta A0". This auxiliary MLS is crucial to permit the transformation of model "X" from "X.B" to "X.A". This is detailed in Fig.12 together with an overview of the whole concept.

Therefore "MLS B.A" has the following features:

- In "MLS B.A" the super-metamodel is the same as the one in "MLS A"
- The model "X" conforms to super-metamodel "Meta B0" in "MLS B" the same way that the same model "X" conforms to "Meta B0" in "MLS B.A". However, as Fig.12 indicates, "Meta B0" in "MLS B.A" also conforms to "Meta A0", instead in "MLS B" the model "Meta B0" is a super-metamodel therefore conforms to itself.

After having defined the two top layers in "MLS B.A", the actual model has to conform to them. Thus we have to pass from a model conforming to the initial MLS to a model conforming to auxiliary MLS.

Getting back to our example at Fig.12, we still need a representation of the model "X.B" conforming to the new "MLS B.A" meta-layers.

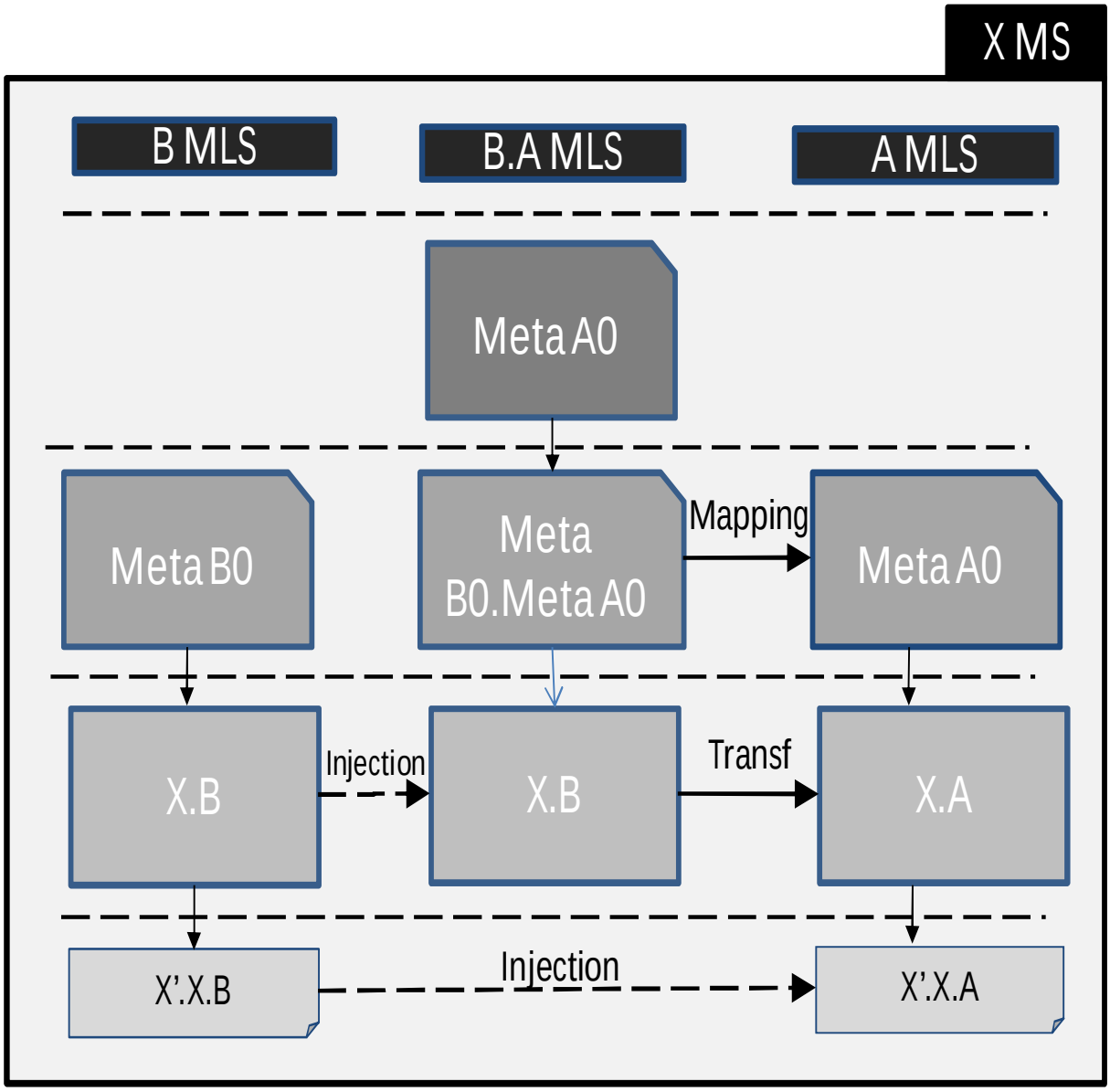

FIGURE 12 - SHOWS "MS X" CONTAINING "MLS B" AND "MLS C", AND THE GENERATED "MLS A" THROUGH MODEL TRANSFORMATION

However it is not possible to pass from "MLS B" to "MLS B.A" in a model driven approach since both super-metamodels are heterogeneous. Therefore, it is used an auxiliary tool to inject the "X.B" model from "MLS B" into a "X.B" model from "MLS B.A". An injector is a tool that grabs a model in its original language and transforms it conforming to the desired new language. But, as the scope of the thesis is all about model-driven approaches we will not describe this passage further.

It deserves being said that although this particular example has an input MLS of three layers, if instead it had a four layered MLS the way to solve it would have been exactly the same, using intermediate MLS to make the link between the two heterogeneous MLS.

Since "MLS B.A" and "MLS A" share the same super-metamodel ("Meta A0"), a model transformation is used to transform the model "X.B" as shown in Fig.12. The transformation output is the same model "X" but this time conforming to "Meta A0" directly.

By now our "MLS A" is almost created, only missing the data layer. Again, since the passage from "X'.X.B" data to "X'.X.A" data can't be done through a model-driven approach it is used an auxiliary tool to inject "X'.X.B" into "X'.X.A". This passage is out of the scope of the thesis, therefore we will not give further details.

Therefore Fig.12 deals with two main issues. The first is how the "MLS B" can be mapped into "MLS A". The second issue is the fact that some MLS can't describe data at all, and a good example of it is the "MLS B.A".

The same whole process should have been done to "MS Y" if it were not already described in "MLS A".

Executing the final Model Transformations

The last step is the simplest when compared to the route that took us here, but is actually the one that enables the completion of the previously defined goal of exchanging data models.

The two Modelling Spaces finally share a common MLS, the "Meta A0", and are therefore ready to execute a transformation between them. This step is the same described in section 4.1, and is represented in the Figure 13 below.

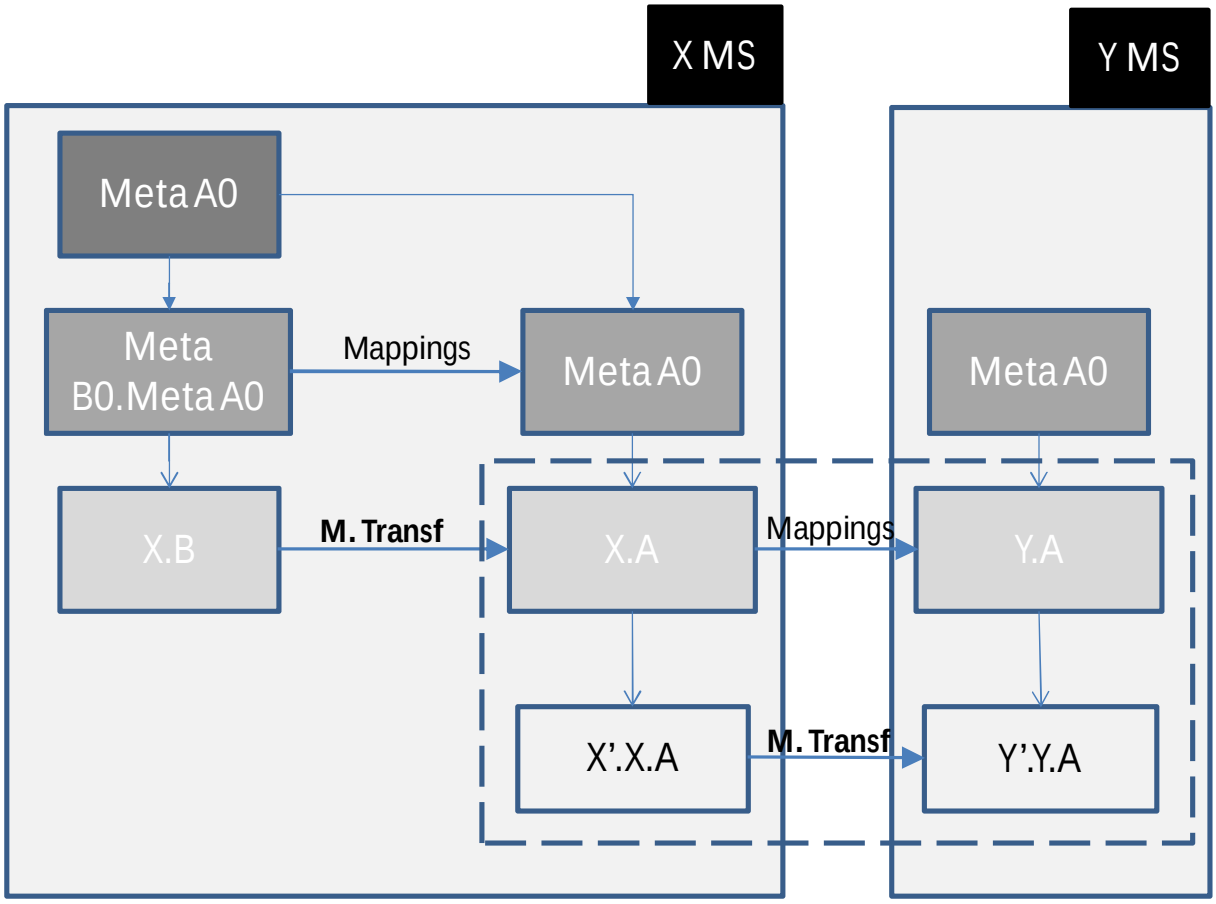

FIGURE 13 – MODEL TTRANSFORMATION REPRESENTING THE LAST STEP

The MS using the model "X" can lastly transform information to the other MS based on model "Y".

It is worth noticing that the reverse path can be done too. This means that if the "MLS A" wasn't already the "MS Y" original format, then the reverse transformation from "MLS A" to the hypothetical original MLS would have to be executed too.

4.3. SUMMARY

This section describes how to use the theoretical concepts established in the previous chapters to implement a solution to solve the issued problem.

That way, directions were given on how to exchange data between two models in heterogeneous Modelling Spaces under two specific situations.

Firstly, whenever the two Modelling Spaces shared a common MLS between them. In that case a Model Transformation is executed and no additional steps are required.

Lastly, whenever the two Modelling Spaces did not share a common MLS between them. In those situations are required three additional steps: First, defining a common MLS; Second, transforming the original MLS into the target common MLS defined in the previous step; Third, executing the actual model transformation between the heterogeneous models from different Modelling Spaces.

5. SOLUTION ANALYSIS AND VALIDATION

In the previous chapter was defined a solution, with the help of model transformations, to cope with the heterogeneity in models. In order to validate and analyze the proposed solution it has to be developed an implementation. Therefore in this section is presented all-together a reference implementation, and a description of the testing scenario. From the testing scenario derive some tests that will help formulating a verdict regarding the presented implementation.

5.1. APPROACH

This chapter was divided in three important topics: Test Methodology, Test notation and Proof of concept. Referring to the main topics for the solution analysis and validation.

5.1.1. TESTING METHODOLOGY

Several methodologies exist to test the skills of the solutions to achieve their requirements, therefore each one of them has their own specific application domain (Onofre, 2007). Ideally, it should not be necessary that the same product is tested more than once by different testing laboratories. This is only possible if testing is based on generally accepted principles, using generally accepted tests, and leading to generally accepted results. To achieve that, the International Organization for Standardization (ISO), together with the CCITT, has developed a standard for conformance testing of Open Systems. This is the standard ISO IS-9646: “OSI Conformance Testing Methodology and Framework” [ISO91]

The purpose of this standard is to define the methodology, to provide a framework for specifying test suites, and to define the procedures to be followed during testing, leading to comparability and wide acceptance of test results produced by different test laboratories, and thereby minimizing the need for repeated conformance testing of the same system. The standard does not specify test for specific protocols, but it defines a framework in which such test should be developed, and it gives directions for the execution of such tests. The standard recommends that sets of test, call test suites, be developed and standardized for all standardized protocols.

The Testing Process

In the process of testing three phases are distinguished. They are depicted in Figure 14, together with the activity of protocol implementation.

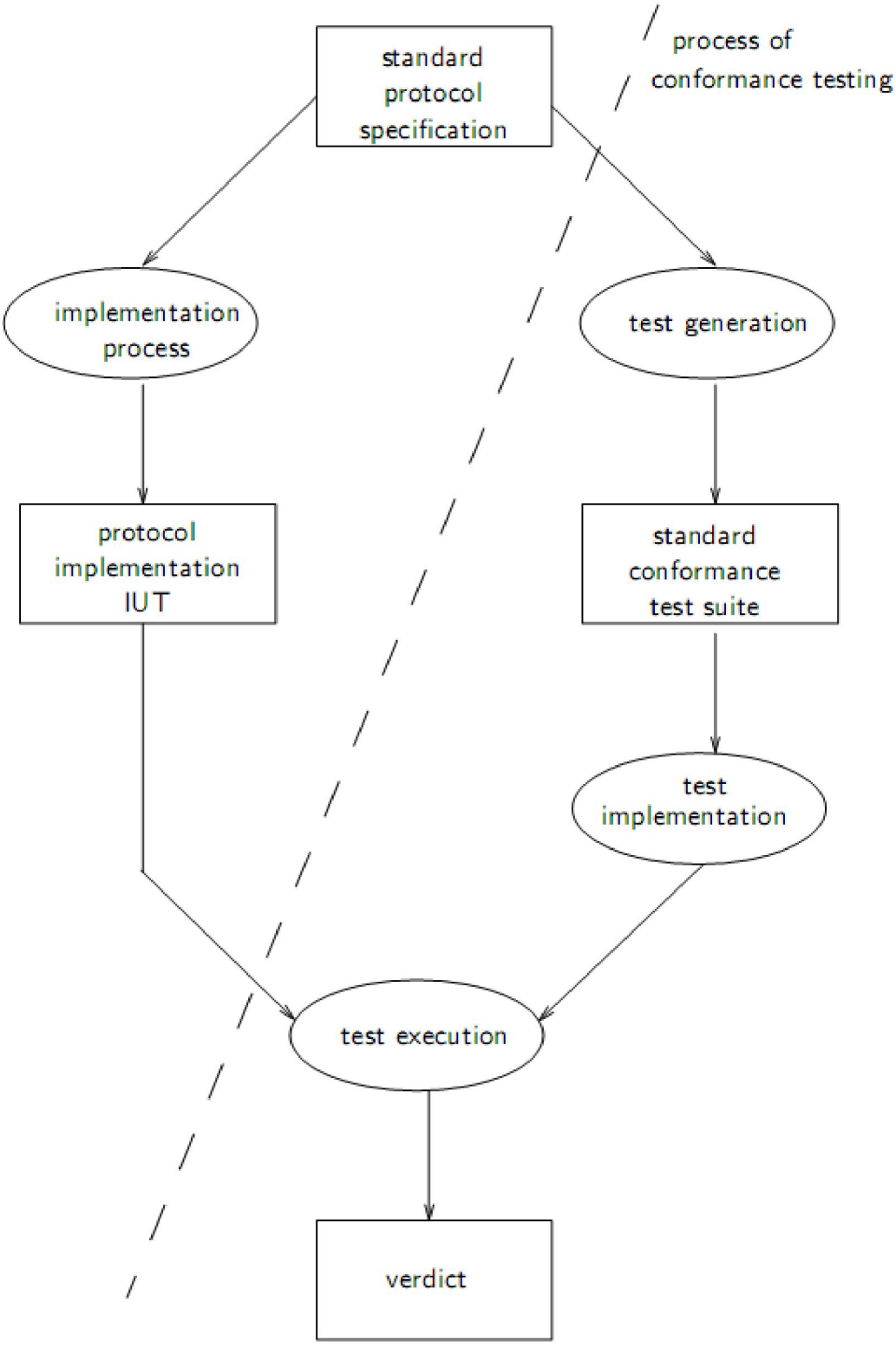

FIGURE 14 – TESTING PROCESS

The first phase is the specification of an abstract test suite for a particular (OSI) protocol. It is referred as test generation or test derivation. This test suite is abstract in the sense that test are developed independently of any implementation. It is intended that abstract test suites of standardized protocols are standardized themselves.

The second phase consists in the realization of the means of executing specific test suites. It is referred to as test implementation. The abstract test cases of the abstract test suite are transformed into executable tests that can be executed or interpreted on a real testing device or test system. The peculiarities of the testing environment and the implementation, which during testing is called IUT (Implementation Under Test), are taken into account.

The third and last phase is the test execution. The implemented test cases are executed with a particular IUT and the resulting behaviour of the IUT is observed. This leads to the assignment of a verdict about conformance of the IUT with respect to the standard protocol specification. The results of the execution are documented in the protocol conformance test report (PCTR).

5.1.2. TEST NOTATION

Since abstract test suites are standardized, they must be specified in a test notation that is well-defined, independent of any implementation, and generally accepted. IS-946 recommends the semi-formal language TTCN, the “Tree and Tabular Combined Notation”, which is defined in [ISO91] and more recently in [ISO97]

This notation follows the black-box testing model: as system is treated as a black box, whose functionality is checked by observing it, i.e., no reference is made to the internal structure of the program and the main goal is to determine whether the right product has been built with respect to the specification.

In TTCN the behaviour of test cases is specified by sequences of input and output events that occur. A sequence can have different alternatives, where different subsequent behaviours can be chosen, e.g., depending on output produced by the IUT. Successive events are indicated by increasing the level of indentation, alternative events have the same indentation. A sequence ends with the specification of the verdict that is assigned when the execution of the sequence terminates.

The events are divided in two types: Actions and Questions. The actions, which are represented with an exclamation mark at the beginning of the event, define the interactions with the system. The questions, which are represented with a question mark, are the expected answers from the system.

The verdict can output three different results: Pass, Fail, or Inconclusive. Pass indicates that the test was executed successfully, and that the goal expressed in the corresponding test purpose was achieved. Fail indicates that the implementation does not conform to the specification. Inconclusive indicates that no evidence of non-conformance was found, but that the test purpose was not achieved.

TABLE 2 – TEST EXAMPLE

Test example	
Test name: Basic connection	
Group:	
Purpose: Check if a phone call can be established	
Comments:	
Behaviour	**Verdict**
! Pick up the phone handset	
? Dialling tone	
! Dial the number	
? Ringing tone	
? connection established	
! Hang up the phone	Success
? Busy tone	
! Hang up the phone	Inconclusive
? No tone	Fail

The test above shows a basic example based on the phone call system with three different verdicts depending on the behaviour of the system.

5.1.3. PROOF OF CONCEPT

Since we are inside the ICT domain, in order to make this tests possible an implementation was made. This implementation is classified under the Technology Readiness Levels. Technology Readiness Level (TRL) is a measure used by many of the world's major companies (and agencies) to assess the maturity of evolving technologies (materials, components, devices, etc.) prior to incorporating that technology into a system or subsystem. Generally speaking, when a new technology is first invented or conceptualized, it is not suitable for immediate application. Instead, new technologies are usually subjected to experimentation, refinement, and increasingly realistic testing. Once the technology is sufficiently proven, it can be incorporated into a system/subsystem.

This implementation in specific is classified under the third level (Proof of Concept Demonstrated, Analytically and/or Experimentally) according to Technology Readiness Levels. That level stipulates that must be included an analytic study to define the appropriate technologies as well as tests in a controlled environment to validate the solution concept proposed.

5.2. TEST DEFINITIONS

As described before in the proposed testing scenario, we will only apply the tests to the solution for the two described scenarios. The first scenario describes Transformations between heterogeneous MS with the same MLS. The second scenario describes Transformations between MS with heterogeneous MLS.

Homogeneous Transformations Test

That way, in the first test we start by defining a source and target Modelling Space. These two modelling spaces must be heterogeneous in the sense that the models that they describe must differ in structural aspects, and at the same time they both share a common MLS describing their models, therefore are homogeneous in the MLS level.

TABLE 3 - HOMOGENEOUS TRANSFORMATION TEST

Test	
Test name:	Homogeneous Transformations
Group:	Homogeneous Transformations Group
Purpose:	Test the proposed guidelines
Comments:	Same MLS

Behaviour	Verdict
! Identify the source and target Modelling Spaces	
? Heterogeneous MS	
! Identify MLS	
? Homogeneous MLS	
! Define instance mapping rules	
! Execute transformation	
? Valid information in target model	Success
? Invalid target model	Fail
? Heterogeneous MLS	Fail
? Homogenous Modelling Spaces	Fail

Next comes the definition of the rules that will map both models and therefore enable the execution of the transformation itself. Since we are dealing with a transformation between the same MLS, the resulting outputted model has to be identical with the source model only differing in structural aspects.

Heterogeneous Transformation Test

In the second test, the Modelling Spaces are chosen so that they have heterogeneous MLS describing each of their models. The structure of each model is similar although described in different MLS. That way is required to define a common MLS between the two Modelling Spaces, and the source model has to be transformed into a representation of the common MLS. Next the rules to transform the models are defined and the two Modelling Spaces can execute the transformation.

TABLE 4 – HETEROGENEOUS TRANSFORMATION TEST

Test		
Test name:	Heterogeneous Transformations	
Group:	Heterogeneous Transformations Group	
Purpose:	Test the proposed guidelines	
Comments:	The same model in Heterogeneous MLS	
Behaviour		**Verdict**
! Identify the source and target Modelling Spaces		
? Heterogeneous MS		
! Identify MLS		
? Heterogeneous MLS		
! Define a common MLS between MS		
? Common MLS defined		
! Transform each MLS into the chosen common MLS		
! Define transformation rules		
! Execute the final model transformation		
? Same information in source and target models		Success
? Source and target models differ		Fail
? No common MLS possible		Fail
? Homogenous MLS		Fail
? Homogenous Modelling Spaces		Fail

After the transformation is executed, the target model can be validated through a comparison with the source model.

5.3. IMPLEMENTATION

To implement the two tests defined in the previous section, we started by creating the source and target model. The chosen heterogeneous models for the implementation are described in UML and EXPRESS and both, and are simple descriptions of a model Book (name, author, and year) though differing at some points to avoid a direct one-to-one transformation as in the case of the Homogeneous Transformation Test.

Unified Modelling Language (UML) (OMG, 2003), was created by the Object Management Group (OMG), and is a standardized general-purpose modelling language in the field of software engineering. UML helps to specify, visualize, and document models of software

systems, including their structure and design, in a way that meets all of these requirements. With its rich palette and middleware independence, UML forms a foundation of OMG's Model Driven Architecture (MDA).

The information modelling language used throughout STEP and other international standards is EXPRESS. The EXPRESS language, defined in ISO 10303 Part 11 (ISO10303-11), is a data modelling language that provides a fairly rich set of facilities for defining complex data types. It is based on the entity-attribute modelling approach.

The part21 (ISO10303-21)- also called STEP Physical File - is the most widely used exchange format throughout STEP. Due to its ASCII structure it is easy to read with typically one instance per line. Although EXPRESS and P21 are directly included in STEP, they don't relate directly. EXPRESS (Part 11) describes models and P21 represents data. In other words, EXPRESS models conform to EXPRESS metamodel and p21 data conforms to P21 metamodel.

Instead of using Part21, Part 11 could also be used to instantiate data, however that would bring some problems since the model and the actual instances would have to be in the same file. That is obviously unpractical, therefore to describe data conforming to EXPRESS models we used P21.

As discussed in chapter 3 (three), the solution will be implemented with ATL M2M project from Eclipse framework. Firstly, an ATL project will be created to support the necessary transformations and since ATL supports runtime execution, it will generate an "*.asm" file each time it compiles the transformations. This is an assembly file that the ATL Virtual Machine can interpret, and thus enable to run the transformations embedded in other Java projects which will only have to define the source and target models and metamodels.

Since ATL works with Ecore as reference for input models, and in order for EXPRESS files to be executed in ATL they have to be parsed into Ecore files. This is done with use of a non-model driven tool that is currently being developed as a parallel work inside the research group, which takes as input an EXPRESS schema and outputs an Ecore model representing the inputted schema.

5.4. EXECUTION

To validate both tests we will and since the first test, Homogeneous Transformations Test, is contained in the execution of the second, Heterogeneous Transformation Test, we chose to present a reference execution that will enable to verify the validity of both tests.

The main goal is to transform data models from "MS eBook" (in "MLS EXP.WSN") to data models from "MS uBook" (in "MLS MOF.UML") using for that purpose ATL transformation engine.

The Fig.15 represents a simplified view from each model Modelling Space, since it only includes the initial MLS that is describing the models.

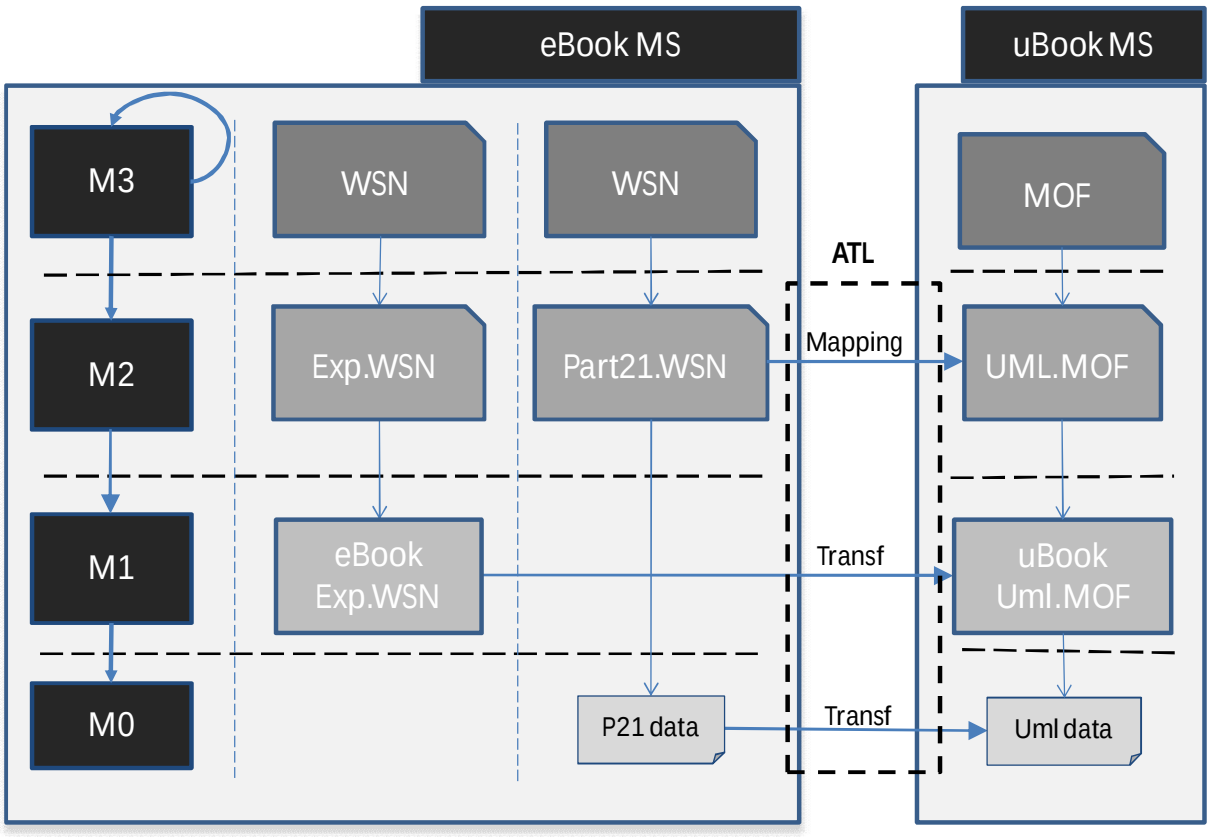

FIGURE 15 - REPRESENTS A SIMPLIFIED VIEW FROM THE MODELLING SPACES OF MODELS "UBOOK" AND "EBOOK"

Analyzing the Fig.15 we can conclude that we are in the presence of two MS described in different MLS, just like it was described in the section 3.2 above. We will then follow the steps described there.

The first step solves the problem that both metamodels conform to different meta-metamodels, (Express conforms to WSN and UML conforms to MOF) and so there is no common base. Analyzing the complete Modelling Spaces in Fig.16 and Fig.17, and since EXPRESS can both be described in WSN and Ecore, and UML representations can described either in Ecore or MOF, we will use Ecore as the common base in the two Modelling Spaces.

The second step deals with the passage from the current MLS in which models are described to the chosen common Ecore MLS.

We have already UML and EXPRESS metamodels described by Ecore. Thus, we transform the models (uBook and eBook) at layer M1 in Fig.15 to make them conform to the new language that is Ecore.

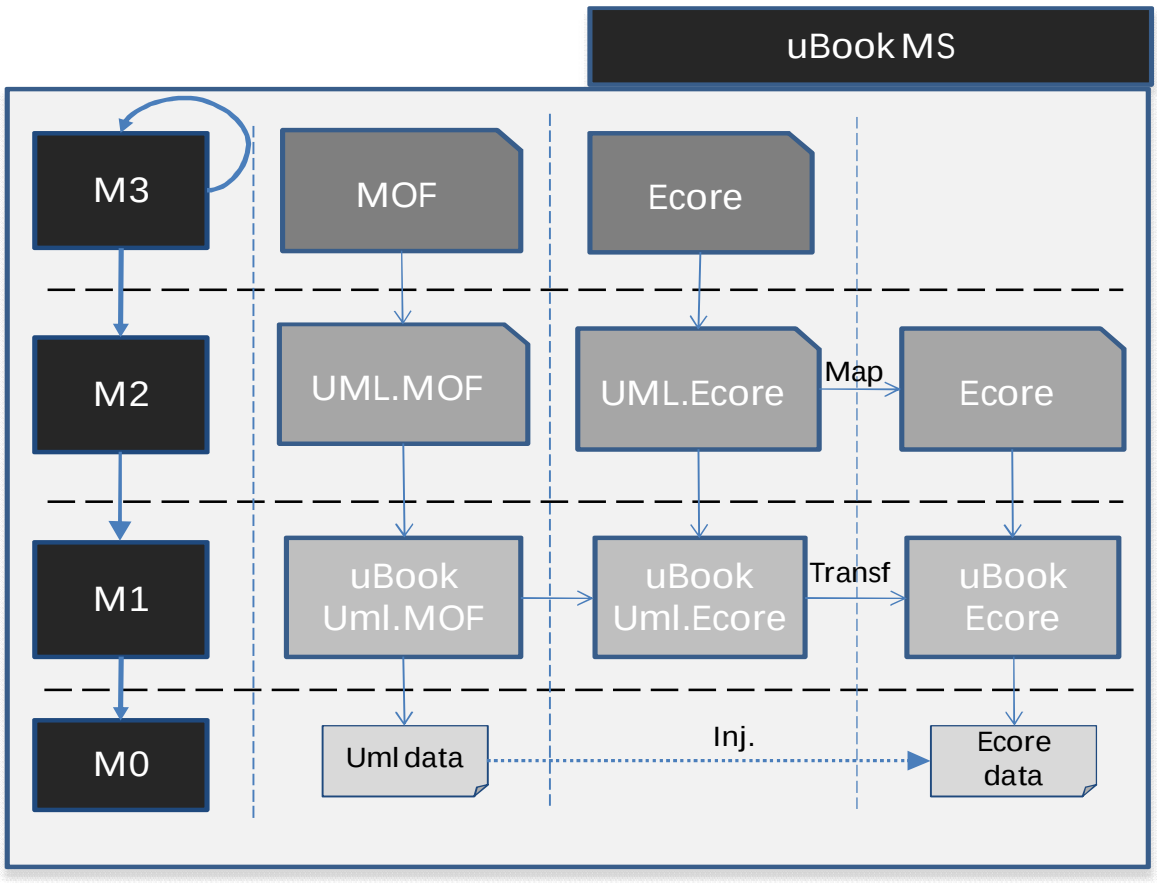

FIGURE 16 - UBOOK MODELLING SPACE

To make the passage from an UML model conforming to the UML metamodel described in MOF, to an UML model conforming to the UML metamodel described in Ecore, we used an

injector that generates the “uBook UML.Ecore”. The same is made for the EXPRESS side, outputting “eBook Exp.Ecore”.

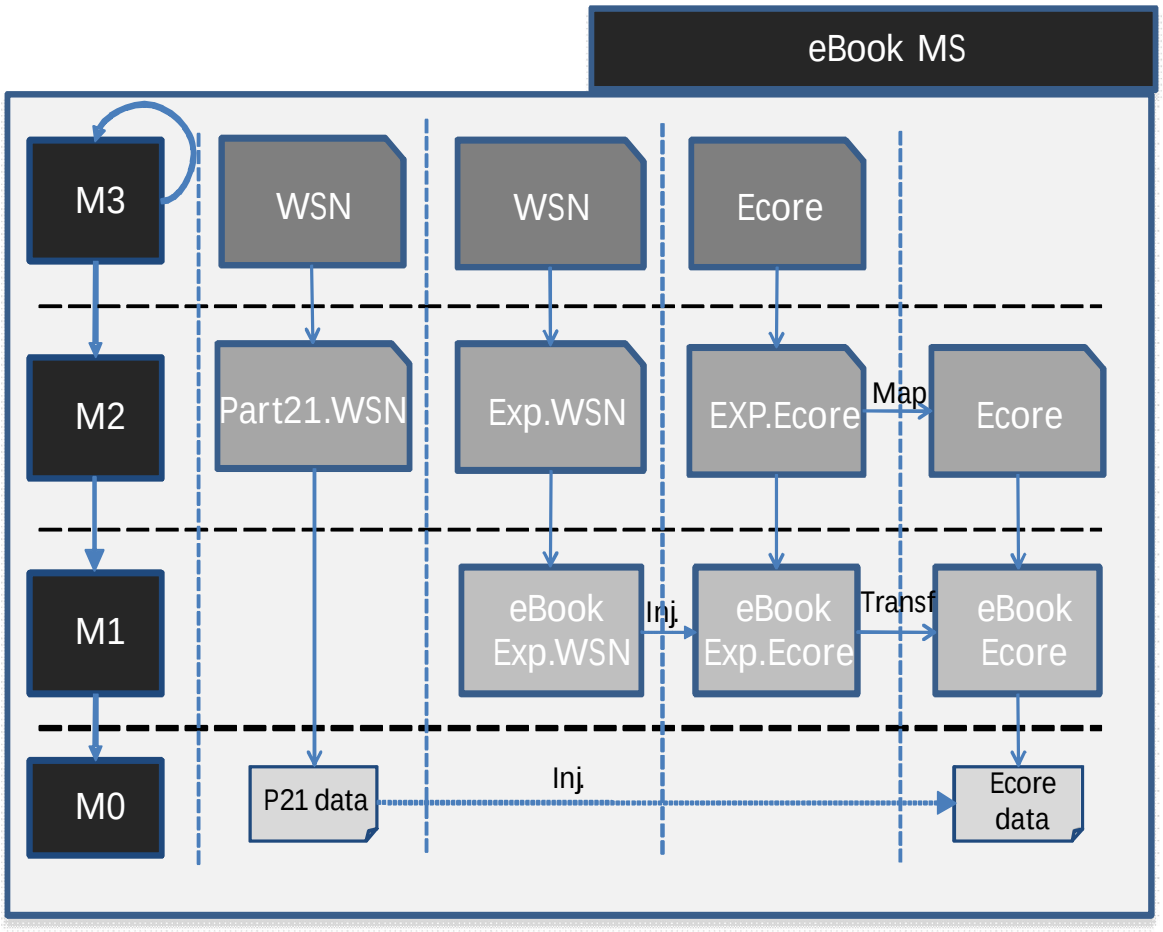

FIGURE 17 - EBOOK MODELLING SPACE

Like we said before, EXPRESS models, like eBook, use STEP Part21 to describe data. This issue is related in Fig.17. Since EXPRESS uses a different MLS only to describe his data in P21 we will treat it independently. Although this is relevant information, we will only approach this subject later in the thesis.

At the end of this step comes the common part between the two specified tests in sub-chapter 5.2. Therefore, until now, this execution was only analyzing one tests, and from now on it will be validating both tests.

To sum things up, after the first step is completed we have:

- The model “eBook” described under the “MLS EXP.Ecore”
- The model “uBooks” described under “MLS UML.Ecore”

However, our goal is to transform the initial P21 “eData” to “uData” and both these models (“eBook” and “uBook”) cannot describe data yet. Therefore we lower Ecore from level M3 to level M2, and we transform our “eBook Exp.Ecore” that was until now described in Express under Ecore, into an “eBook.Ecore” described directly in Ecore. The same is done in UML side to achieve a model “uBook.Ecore”. Both these new models still represent real world models (Book, Persons, etc.) to whom data will conform to.

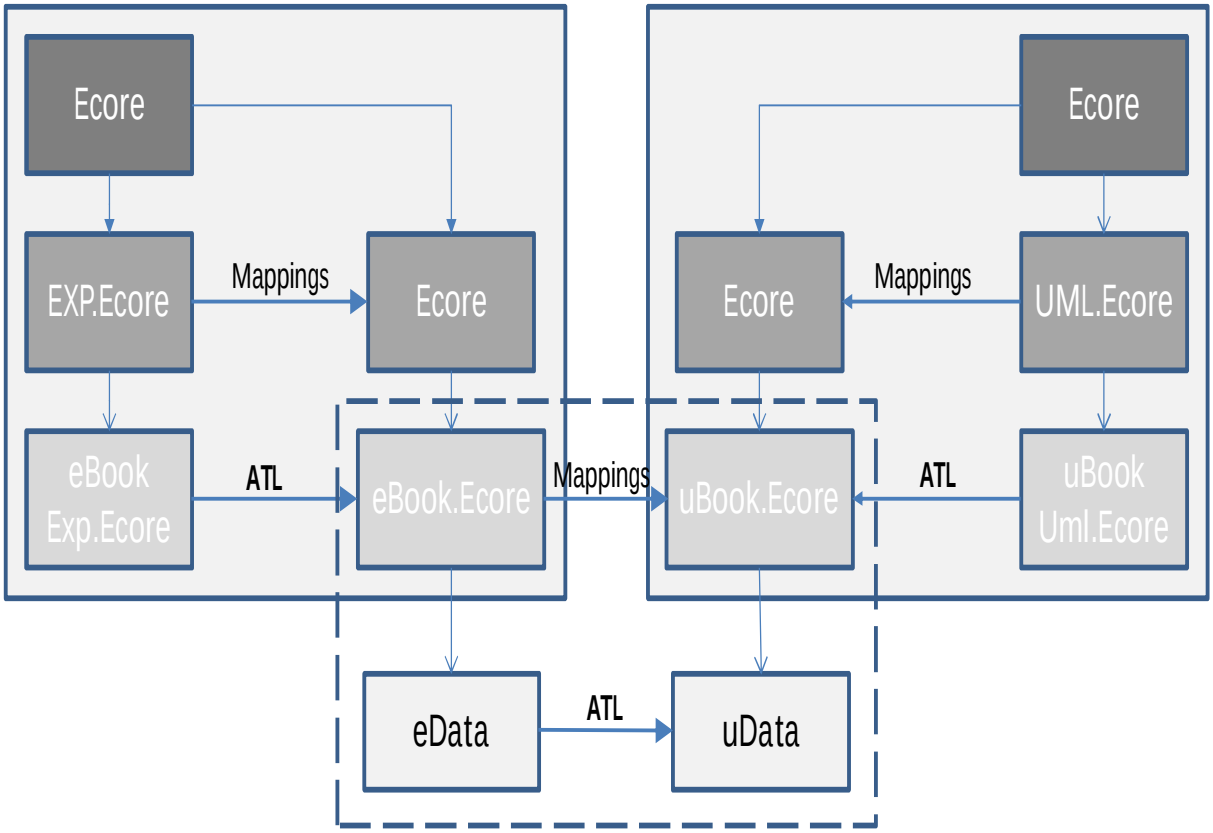

FIGURE 18 - PROCESS TO TRANSFORM EBOOK DATA IN UBOOK DATA, REPRESENTING THREE DIFFERENT ATL TRANSFORMATIONS

Until now, the initial “P21 Data” described under “MLS P21.WSN” remains unchanged. Therefore, to finally handle it, we transform it into data conforming to the model “eBook.Ecore”. Given that this in only attainable with the help of existing non-model-driven tools, is considered out of the scope of the thesis.

By now, we already have every model described in Fig.18 created and therefore ready for the ATL code execution. This transformation is identical to the one described in section “3.1” since it involves two Modelling Spaces with a common MLS that is in this case “Ecore”.

The opposite work could also have been done. In other words, the transformation of Uml data models into P21 data models is inside the scope of the thesis. To create an “uData” model, it would have to be done the exact same way it is done to create an “eData” model conforming to Ecore. But again, since this is too only affordable with the help of existing non-model-driven tools, is considered out of the scope of the thesis.

The execution outputted the expected data model “uData”. This data model could be transformed back to an UML data representation.

5.5. VERDICT

The conclusion from the tests is that the outputted files are identical to the expected ones, meaning that the proposed Guidelines are valid.

Having in mind that we are in presence of a proof-of-concept, we can define as a final verdict about the solution facing its objectives, and although the tests were made with a simple source EXPRESS “schema” it is relevant enough to call it successful. However, it would be important to execute other tests with more complexity in order to understand how the implementation would respond. Another point worth noticing is the fact that the tests were only executed with EXPRESS models as input, therefore would be important to execute the tests with other formats.

Thus, we can conclude that since the problems faced during development were directly connected with technical issues or due to technology bugs, and not conceptually, which was the prime objective of the analysis, the solution is valid.

5.6. SUMMARY

This chapter describes the methodology, notation and the Proof of concept used to analyse and validate the proposed solution. The tests were defined contemplating the solution for the two described scenarios. The first scenario describes Transformations between heterogeneous MS with the same MLS. The second scenario describes Transformations between MS with heterogeneous MLS.

After the tests were defined, a Reference Implementation is suggested as well as some execution tests. To validate both tests and since the first test, Homogeneous Transformations Test, is contained in the execution of the second, Heterogeneous Transformation Test, we chose to present a reference execution that will enable to verify the validity of both tests together. These processes altogether helped reaching a verdict concerning the validity of the proposed solution.

6. CONCLUSIONS AND FUTURE WORK

To cope with the constant change of demands from the business conjuncture, to improve competitiveness and agility, companies are coming together within Collaborative Networks in order to be able to cooperate dynamically and offer complex services, create new market opportunities, combine their knowledge, products and services, and jointly produce and offer new services and products.

The competitiveness and success of such networks is highly determined by the ability which their members have to interoperate so that all members within a network can behave as a single enterprise. However, due to the diversity and most especially to the heterogeneity of systems, applications and technologies used among CNs members, results that information from different CN members end up differing at many levels, causing several interoperability problems.

In that sense, Model Driven Development based on Model Transformations has proven to be a critical foundation in granting an efficient way to overcome these interoperability problems, and leads to advantages such as flexibility, capability of changing with ease and to obtain a general view over the whole system. In order to take advantage of all these model transformations features, a transformation tool must be used.

However, and despite the fact that many important model transformation tools already exist, none is actually capable of directly and efficiently providing the ability to execute transformations between models described under two distinct and heterogeneous model concepts. This is mainly due to technical limitations in both these tools and transformation languages that these tools support and sometimes even the modelling languages in which two models are described are so different one from another in their meta layers, that makes it impossible to directly execute transformations between the heterogeneous models.

Therefore, the main motivation for this work is the fact that, regardless currently there are already several tools offering excellent support to model-to-model (M2M) transformations, has to be developed a way to execute model transformations between heterogeneous

model concepts with the use of tools more dedicated at executing transformations between heterogeneous data models.

To this end, the state-of-the-art regarding Transformation Environments was studied in order to identify which Transformation Language and Transformation Engine would mostly fit our requirements. As a result of the study, where points like License, Supporting Community, Simplicity, among other were taken into account, ATL emerged as the TE that most fits our needs.

A concept guideline was proposed based on the concepts of Modelling Spaces which is a modelling architecture representing a specific model that can be described in various different modelling languages, and each modelling language that describes models has a Meta Language Space. With the use of Model Driven Development basic operation - Model Transformations - these guidelines take advantage of all the current features of the existing transformation tools in order to provide a solution to the problem of executing transformations between models described in heterogeneous model concepts. In general that solution is accomplished with means of three steps: Defining a common MLS between MS; Transforming each Modelling Space MLS to the chosen common MLS; Executing the final model transformation as a regular transformation between models described in homogeneous MLS;

An analysis of how the solution could be realised has been presented where different options regarding the implementation and technical aspects were discussed. In addition, the implementation of the solution has been validated through the development of a series of tests. The development of the tests followed the analyses mentioned above, and it was described in detail, explaining the major design and implementation decisions. The tests successfully validated the solution presented with this work.

As a final conclusion, and concerning the results, a direct illustrative contribution of this work is to help reduce the complexity, costs and the barriers when enterprises, working with heterogeneous Modelling Spaces inside Collaborative Networks, want to communicate between them. This contribution is especially relevant for the SMEs, since this way they can reduce the costs of engaging in strategic business alliances, hence increasing their competitiveness, by reducing major interoperability costs that hinder business performance today.

As for future works, a good starting point would be to continue the development of this work in some particular steps, that couldn't be done with a model driven approach, being required the use of some hard coded injectors to do the job. This is due to technical limitations in the model transformation tools, and it is expected that in a short period of time some of these limitations will be suppressed with the launching of new software versions or updates. That way, and completing the additional and the unfinished steps, the approach would then become based fully on model driven development.

Additionally, the current work only supports EXPRESS and UML Meta Language Spaces, therefore, it could be of great use to implement more transformations rules so that other MLS could be supported, and thus increasing the interest of developers and compatibility among different systems.

From a scientific perspective, a future research challenge that is envisioned is to develop a full interoperability framework for models in general, thus extending and evolving the solution here proposed, such as Model Servers (KAJ, 2008) that are frameworks that support persistence of models, by which multiple users can share models and execute operations in them. For example, the solution should be able to combine, associate, link, and translate models. Moreover, besides model management and processing, the envisioned Model Server should target also the execution of models. By execution is meant, the capability to process data structured according to the models. Also it should allow different models to communicate (i.e. to exchange data) in an "ad-hoc" manner, without using a common neutral format as data backbone. This entails that models should encompass their own behaviour description and also, introspection mechanisms must be available to allow them to adapt and adjust "on the fly".

Such interoperability framework would allow the development of highly dynamic and adaptive systems that could accommodate unforeseen changes in their operation environments and react appropriately with few or even no human intervention, meaning it should, for instance, support the dynamic generation of model transformations. In other words, one should be able to generate transformation rules from point "A" to "C" if the "path" from "A" to "B" and "B" to "C" is known. Nevertheless, the creation of such solution in not straightforward, since it should provide advanced capabilities, such as the automatic

discovery of properties and inference mechanisms to support intelligent decision making, hence it presents some interesting research challenges.

Some of these scientific contributions are currently being developed, which already show reasonable results, by colleagues in the research group GRIS-UNINOVA, in which the author is currently working.

It is worth noticing that the result of the work being developed with this thesis was used in an industrial project finances with QREN - *Quadro de Referência Estratégico Nacional* – and named Building Studio, which is currently being promoted by CXS Computing S.A.

(http://www.porlisboa.qren.pt/np4/file/130/lista_beneficiarios.pdf).

A scientific article was published as a direct result of this work, with the title “Directives for applying model-driven transformations to heterogeneous models and modelling languages” which in the date of delivery of this document is in the reviewing phase of submission in Models 2010.

Additionally a local internal workshop, on the thematic of the current thesis, will be coordinated by the author with the objective of stimulating and encouraging new Msc students to approach into the Model Driven Interoperability world.

7. BIBLIOGRAPHY

[Online] // Xmorph. - http://xmorph.sourceforge.net/.

Budinsky F. [et al.] Eclipse Modeling Framework, Chapter 5 "Ecore Modeling Concepts" [Book]. - [s.l.] : Addison Wesley Professional, 2004. - ISBN: 0131425420.

Camarinha-Matos Luis M. and Afsarmanesh Hamideh Collaborative networks: a new scientific discipline [Journal] // Journal of Intelligent Manufacturing. - 2005. - Vol. 16. - pp. 439–452.

Camarinha-Matos Luis M. and Afsarmanesh Hamideh COLLABORATIVE NETWORKS: Value creation in a knowledge society [Conference] // PRogramming LAnguages for MAchines Tools (PROLAMAT) conference / ed. Springer. - Shangai, China : [s.n.], 2006.

Chen David, Dassisti Michele and Elvesæter Brian Enterprise Interoperability Framework and knowledge corpus - Final Report, Deliverable DI.3 [Report]. - [s.l.] : InteOP-NoE - EU research project (IST-508 011), 2007.

Cook Steve Domain-Specific Modeling and Model Driven Architecture - The Role of Models in Software Development [Journal] // MDA Journal. - 2004.

Djuric D., Gasevic D. and Devedzic V. The Tao of Modelling Spaces [Journal]. - [s.l.] : Journal of Object Technology, 2006. - Vol. vol 5 no 8.

EMF [Online]. - http://www.eclipse.org/modeling/emf/.

EPL [Online]. - http://www.eclipse.org/legal/epl-v10.html.

Eriksson Anna, Feizabadi Ali and Zamani Baranoush Integration of Heterogeneous Systems [Book]. - 2001.

Gray John Engineering supply chain data exchange [Article] // International TechneGroup Incorporated. - 2003.

GREAT [Online]. - http://www.isis.vanderbilt.edu/tools/GReAT.

Gunasekarana Angappa, Laib Kee-hung and Chengb T.C. Edwin Responsive supply chain:A competitive strategy in a networked economy [Journal] // The International Journal of Management Science. - 2008. - 4 : Vol. 36. - pp. 549 – 564.

Hailpern B. and Tarr P. Model-driven development: The good, the bad, and the ugly [Journal] // IBM SYSTEMS JOURNAL. - 2006. - 3 : Vol. 45.

Hausmann Jan Hendrik, Heckel Reiko and Lohmann Marc Model-based Discovery of Web Services [Conference] // Proceedings of IEEE International Conference on Web Services. - 2004. - pp. 324- 331. - ISBN: 0-7695-2167-3.

Hidaka S., Zhenjiang H. and Kato H. A compositional Approach to Bidirectional Model Transformation [Journal]. - 2008.

ikv MediniQVT [Online]. - http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77.

Irazabal Jeronimo, Pons Claudia and Neil Carlos Model transformation as a mechanism for the implmentation languages [Journal]. - Buenos Aires : [s.n.].

ISO10303-11 Product data representation and exchange: Description method: EXPRESS Language Reference Manual - Edition 2 // Industrial Automation and Integration- Product Data Representation and Exchange - Part 11.

ISO10303-21 STEP Part 21, Implementation method: Clear Text Encoding of the Exchange Structure // Industrial Automation and Integration- Product Data Representation and Exchange - Part 21.

Jouault F. and Kurtev I. On the Architectural Alignment of ATL and QVT [Journal]. - Nantes : [s.n.].

KAJ JS, PC, KS, KBS, JM Use of IFC Model Servers [Book]. - 2008.

Kim Hyun [et al.] A framework for sharing product information across enterprises [Journal] // The International Journal of Advanced Manufacturing Technology. - [s.l.] : Springer, January 2006. - Vol. 27. - pp. 610-618. - 5-6.

Kleppe Anneke G., Warmer Jos B. and Bast Wim MDA Explained: The Model Driven Architecture : Practice and Promise [Book]. - [s.l.] : Addison-Wesley Object Technology, 2003. - ISBN-13: 978-0321194428.

Kurtev Ivan Adaptability of Model Transformations [Report]. - 2005.

Kutsche R.-D.,S¸nb¸l,A. A Meta-Data Based Development Strategy for Heterogeneous, Distributed Information Systems [Conference] // Proc.3rd IEEE Metadata Conference. - Bethesda, Maryland : IEEE Computer Society, 1999. - Vols. April 6-7 1999.

Lee Hau L. and Whang Seungjin E-Business and Supply Chain Integration [Article] // Stanford Global Supply Chain Management Forum, Stanford University. - 2001.

LI Hai-yue [et al.] A Web-based PLM System Research and Implementation in a Collaborative Product Development Environment [Conference] // IEEE International Conference on e-Business Engineering (ICEBE). - 2005. - pp. 549-552.

Lin Yuehua, Zhang Jing and Gray Jeff Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development [Article] // OOPSLA & GPCE Workshop Best Practices for Model Driven Software Development (OOSPLA'04). - Vancouver : [s.n.], 2004.

Mens Tom, Czarnecki Krzysztof and Gorp Pieter Van A Taxonomy of Model Transformations [Article] // Language Engineering for Model-Driven Software Development - seminar. - 2005.

Miles R.E. and Snow C.C. The new network firm:a spherical structure built on a human investment philosofy [Journal] // Organizational Dynamics Journal. - 1995. - Vol. 23. - pp. 5-18.

Myhr Niklas and Spekman Robert E. Collaborative supply-chain partnerships built upon trust and electronically mediated exchange [Journal] // Journal of Business & Industrial Marketing. - 2005. - 4 : Vol. 20. - pp. 179-186.

Nachira Francesco, Dini Paolo and Nicolai Andrea [Online] = A Network of Digital Business Ecosystems for Europe: Roots, Processes and Perspectives // Technologies for Digital Ecosystems - Innovation Ecosystems Initiative. - 2006. - January 2008. - www.digital-ecosystems.org/doc/papers/d5-intro-for-the-press.pdf.

Nordmoen Bjørn BEYOND CORBA MODEL DRIVEN DEVELOPMENT [Online] // MDA. - 2001. - January 2008. - http://www.omg.org/mda/mda_files/SSSummit_nordmoen_OMG.pdf.

OMG CORBA [Online]. - http://www.corba.org/.

OMG UML 2.0 [Online]. - 2003. - http://www.omg.org/docs/ptc/03-08-02.pdf.

OMG's MetaObject Facility (MOF) Home Page [Online]. - March 2008. - http://www.omg.org/mof/.

Onofre Sérgio Miguel da Silva Plataforma para testes de conformidade de sistemas baseados em módulos conceptuais STEP [Report] : Dissertação de Mestrado / Departamento de Engenharia Electrotécnica ; Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia. - 2007.

Park Jinsoo and Ram Sudha Information Systems Interoperability: What Lies Beneath? [Conference] // ACM Transactions on Information Systems. - 2004. - Vol. 22. - pp. 595–632.

Puder A., Pavle G. and Todtenhofer R. Using QVT for Byte Code-Level Cross Compilation [Journal].

Richard Lemesle Transformation Rules Based on Meta-modeling [Conference] // Proceedings of Enterprise Distributed Object Computing Workshop (EDOC'98). - La Jolla, CA, USA : [s.n.], 1998. - pp. 113-122. - ISBN: 0-7803-5158-4.

Schafersman Steven D. Scientific Thinking and the Scientific Method [Online] // An Introduction to Science. - January 1994. - 10 September 2008. - http://www.freeinquiry.com/intro-to-sci.html.

Schmidt Douglas C. Model Driven Engineering [Article]. - [s.l.] : Published by the IEEE Computer Society, 2006. - 0018-9162/06/$20.00 © 2006 IEEE.

Seidewitz Ed What do models mean? [Article] // Software, IEEE. - 2003. - 5 : Vol. 20. - pp. 26-32.

SmartQVT [Online]. - http://sourceforge.net/projects/smartqvt/.

Stevens Perdita Bidirectional model transformations [Journal]. - Edinburgh : [s.n.], 2007.

TEFKAT [Online]. - http://tefkat.sourceforge.net/.

Wimmer Manuel From mining to Mapping and Roundtrip Transformations [Book]. - Vienna : [s.n.], 2008.